UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

SISTEMA EMPOTRADO DE PROCESAMIENTO DE IMAGEN BASADO EN FPGA

Sergio Izquierdo Díaz
Tutor: Gustavo Sutter Capristo

JULIO 2018

SISTEMA EMPOTRADO DE PROCESAMIENTO DE IMAGEN BASADO EN FPGA

AUTOR: Sergio Izquierdo Díaz
TUTOR: Gustavo Sutter Capristo

High Performance Computing and Networking (HPCN-UAM)
http://www.hpcn-uam.es
Dpto. de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2018

Resumen

Hoy en día, el mundo de los sistemas empotrados y su complejidad funcional constituye un inmenso campo de estudio en pleno crecimiento, planteando nuevas plataformas, aplicaciones y dispositivos aún por desarrollar y explorar su impacto en diferentes áreas.

En este Trabajo Fin de Grado se estima y valora la capacidad que tiene el entorno de desarrollo de sistemas empotrados *SDSoC* de dispositivos lógicos programables de Xilinx, con el objetivo de diseñar aceleradores hardware para procesadores ARM. Estos procesadores se encuentran en distintos dispositivos de desarrollo FPGA *Zynq* y *Zynq UltraScale+*. Más específicamente, se evaluará su utilización para originar herramientas y aplicaciones con procesamiento de imagen y vídeo.

La tarea de diseño de estos sistemas empotrados conlleva una alta carga computacional debido a la electrónica implementada que requiere ser dinámica y efectiva en coste, cuyo objetivo final es la integración en dispositivos hardware con un tamaño y consumo de potencia reducidos en comparación con las implementaciones software. Estos sistemas son ampliamente utilizados y están presentes en casi todas las infraestructuras tecnológicas que nos rodean, desde la medicina pasando por desarrollos militares hasta la agricultura. Es muy común encontrar un sistema embebido en cualquier aspecto de nuestra vida cotidiana, ya que están diseñados entre otras cosas para cubrir necesidades básicas del día a día, compuestos por un microprocesador que incluye interfaces de entrada y salida.

Partiendo de funcionalidades ya existentes en la librería xfOpenCV, en la que se encuentran un conjunto de más de 50 kernels optimizados y compatibles con Xilinx para la plataforma *Zybo*, se personalizará el código integrando varias versiones particulares tanto para software como asistidas por hardware, logrando una estimación rápida de recursos, rendimiento y área con comunicación de datos implementable y sintetizable para cada diseño.

Para el diseño con procesamiento de imágenes se ha usado la familia de dispositivos *Zynq*, exactamente la placa de desarrollo *Zybo*. Mientras que el procesamiento de vídeo con una cámara de vídeo se ha llevado a cabo en una nueva placa llamada *Zybo Z7* sacada al mercado hace poco tiempo.

Palabras clave

SDSoC, ARM, FPGA, Zynq, Zynq UltraScale+, Sistemas embebidos, xfOpenCV, Xilinx, Zybo, Zybo Z7, Procesamiento de imagen.

Abstract

Nowadays, the world of embedded systems and their functional complexity is an immense field of study in full growth, raising new platforms, applications and devices yet to be developed and exploring their impact in different areas.

This final degree project will estimate and value the capacity of the *SDSoC* embedded system development environment for Xilinx programmable logic devices, with the aim of designing hardware accelerators for ARM processors. These processors are found in different *Zynq* and *Zynq UltraScale+* FPGA development devices. More specifically, its use will be evaluated to originate tools and applications with image and video processing.

The design task of these embedded systems involves a high computational load due to the computer implementation that requires to be dynamic and cost effective, whose final objective is the integration in hardware devices with a reduced size and power consumption in comparison with the implementations of software. These systems are very useful and are present in almost all the technological infrastructures that surround us, from science to the latest in agriculture. It is very common to find an embedded system in any aspect of our daily life, since they are among other things to cover basic daily needs, consisting of a microprocessor that includes input and output interfaces.

Based on existing functionalities in the xfOpenCV library, in which there is a set of more than 50 optimized and compatible Xilinx kernels for the *Zybo* platform, the code will be customized by integrating several particular versions, both for software and hardware, achieving a rapid estimation of resources, performance and area with data communication implementable and synthesizable for each design.

For the design with image processing the *Zynq* family of devices has been used, exactly the *Zybo* development board. While video processing with a video camera has been carried out on a new board called *Zybo Z7* recently released to the market.

Keywords

SDSoC, ARM, FPGA, Zynq, Zynq UltraScale+, Hibrid systems, xfOpenCV, Xilinx, Zybo, Zybo Z7, Image processing.

Agradecimientos

En primer lugar, quiero dar las gracias a mi tutor, Gustavo Sutter Capristo, por su ayuda e implicación durante la realización de este proyecto.

También por supuesto dar las gracias a mis compañeros del grupo de investigación y como no, a mi familia por su apoyo incondicional durante toda la carrera.

No olvidarme tampoco de los compañeros y amigos que he conocido en la universidad y me han acompañado todos estos años en el grado.

Gracias a todos.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

1 Introducción

1.1 Motivación

En la actualidad, la evolución en el mercado de los sistemas embebidos está teniendo gran importancia en el desarrollo de nuevas plataformas y placas de diseño como es *SDSoC* [16] de Xilinx. Las FPGA se han convertido en la tecnología de implementación de sistemas digitales empotrados más frecuente y extendida de las últimas décadas, ya que su característica fundamental reside en la reconfigurabilidad del dispositivo un número prácticamente ilimitado de veces.

La principal ventaja consiste en que el dispositivo realiza un prototipado previo del circuito obteniendo con ello que el tiempo de diseño y desarrollo se reduzcan notablemente. En este sentido, la simulación funcional durante las primeras etapas del flujo de diseño es una alternativa eficiente para confirmar la adecuada funcionalidad del sistema. No obstante, en sistemas complejos la simulación no es especialmente conveniente debido a los altos periodos de tiempo que puede destinar para ello, por lo que la ejecución hardware es una opción realmente interesante para observar el diseño programable mediante test y pruebas. Por esto mismo, fabricantes de sistemas empotrados como Xilinx no hacen más que sacar al mercado nuevos y potentes diseños software listos para estudiar los entornos de desarrollo y desplegar nuevos horizontes perfeccionando los ya existentes.

De este modo, se está invirtiendo mucho dinero en investigación y progreso de esta tecnología cuyo futuro es muy prometedor, ya que el objetivo es mejorar la productividad del diseño y su optimización.

En este Trabajo Fin de Grado se aborda la metodología de diseño de sistemas empotrados *SDSoC* de Xilinx sintetizando el programa software al hardware requerido para procesamiento de imagen. Todo esto conlleva la previa configuración del kernel y posteriormente la implementación del sistema embebido. A partir de un estudio detallado del rendimiento del sistema, se lleva a cabo su implementación con la finalidad de obtener unas prestaciones aceptables y de bajo consumo, utilizando un lenguaje de síntesis de alto nivel.

Sin embargo, el avance continuo de estas tecnologías y arquitecturas de manera exponencial y acelerada hace que estas plataformas tengan una elevada complejidad en dar soporte por parte del fabricante y eso es un gran inconveniente.

Es por ello que cada vez más se requieren más ingenieros altamente capacitados para investigar y adoptar nuevas metodologías, áreas de desarrollo y herramientas imprescindibles para lograr la depuración de diseños implementados sobre FPGA con recursos de memoria limitados en sistemas software, aunque el objetivo final es llevarlos siempre a casos reales en aplicaciones hardware.

La motivación que ha dado lugar a este Trabajo Fin de Grado ha sido adaptar el software del diseño inicial y desarrollar aplicaciones software con ellas para llevarlas a hardware en la placa de desarrollo *Zybo*, elaborando en cada una de sus mediciones un amplio análisis de rendimiento y recursos empleados para llevarlo a cabo.

1.2 Objetivos

El objetivo de este trabajo es lograr la síntesis e implementación de circuitos partiendo de una versión puramente software de la aplicación. El punto de partida es el uso de la biblioteca xfOpenCV para procesamiento de imagen y vídeo de la herramienta *SDSoC* [16] de Xilinx, que facilita su integración y desarrollo. Asimismo, se completa el estudio con mediciones y análisis de latencias, evaluando la precisión y rendimiento del diseño.

El objetivo inicial será la familiarización con el entorno de desarrollo *SDSoC*, observando las capacidades y limitaciones que presenta para el desarrollo de nuestro proyecto. Esto implicará un profundo estudio previo de la documentación necesaria para entender bien la aplicación y la realización de varios tutoriales y guías básicas con las que adquirir práctica. Es por ello realmente necesario coger cierta destreza para el desarrollo de dichos sistemas, al igual que con la placa de desarrollo Digilent *Zybo* [5] con la que se trabajará a lo largo del proyecto.

El segundo objetivo consistirá en la adaptación de nuestro software a la plataforma de diseño y configuración del kernel del dispositivo que permita llevar a cabo esas funcionalidades. Una vez implementado, se realizará la conexión en el dispositivo de desarrollo a través de una tarjeta micro SD.

Por último, se estudiará los problemas e inconvenientes que hayan surgido durante su implementación para solucionarlos en el futuro y conseguir un progreso en esta tecnología. Esto engloba varios parámetros de diseño como son el consumo de potencia, minimizar el coste en las etapas de diseño y por el contrario aumentar la seguridad del sistema. La intención principal es conocer el flujo de diseño y alcanzar un equilibrio entre tamaño, prestaciones y consumo tanto de energía como recursos, por lo que la plataforma y el diseño hardware del sistema deben ser fundamentales.

Por tanto, este trabajo analítico implica tener ciertos conocimientos en diferentes áreas: arquitectura hardware de procesadores, plataformas y herramientas de desarrollo, dispositivos de desarrollo, procesamiento de imágenes, lenguajes de programación y arquitectura de ordenadores.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Capítulo 1: Introducción. Introducción sobre la importancia del proyecto donde se expone su contexto actual y la necesidad de diseñar sistemas empotrados.
- Capítulo 2: Estado del arte. Se detallará la herramienta de desarrollo utilizada *SDSoC*, así como la metodología de diseño y plataformas usadas.
- Capítulo 3: Procesamiento de imágenes. Descripción de las librerías OpenCV y xfOpenCV usadas para implementación de imágenes, incluyendo las funciones de procesamiento y las imágenes de prueba empleadas.
- Capítulo 4: Procesamiento de vídeo. Descripción de la cámara y demostración de su funcionamiento y procesamiento.
- Capítulo 5: Integración, pruebas y resultados. Presentación de los procesos y pruebas que se han llevado a cabo.
- Capítulo 6: Conclusiones y trabajo futuro. Se exponen las conclusiones sacadas del trabajo realizado.
- Capítulo 7: Bibliografía.

2 Estado del arte

2.1 Entorno de desarrollo SDSoC

2.1.1 Descripción

La herramienta de desarrollo de sistemas empotrados usada en este proyecto para la síntesis e implementación de las distintas versiones de procesamiento es el software *SDSoC*, más específicamente la versión *SDx 2017.2*, corriendo sobre la distribución de CentOS Linux 7.

El entorno de desarrollo *SDSoC* incluye compiladores de aplicaciones C/C++ que producen sistemas completos de software asistidos por hardware dirigidos a dispositivos de desarrollo como *Zynq* y *Zynq UltraScale+* de la compañía Xilinx, encargado de generar desarrollos de sistemas híbridos CPU y FPGA [16]. Además, posibilita la integración de aplicaciones para procesadores basados en ARM que habilitan módulos de aceleración por hardware y análisis clave mientras sintetizan y permiten la implementación de la lógica programable que relaciona la unidad central de procesamiento (CPU) con la FPGA.

SDSoC tiene la capacidad de realizar una estimación del rendimiento del sistema explorando distintos escenarios, obteniendo una evaluación con un análisis de consumo, prestaciones y recursos utilizados de la síntesis. La herramienta genera un proyecto en el que es obligatorio especificar el directorio de trabajo, configuración de sistema, plataforma de desarrollo... Una vez construido e implementado correctamente, el sistema híbrido crea a partir del IP del acelerador, un archivo (.BIN) y un ejecutable (.elf) con el *bitstream* necesario para programar el dispositivo FPGA. Para su conexión y configuración, se utiliza una tarjeta micro SD en la que se introducen estos ficheros.

2.1.2 Metodología de diseño

La metodología de diseño aplicada para sistemas empotrados con la herramienta *SDSoC* plantea el desarrollo tanto software como hardware de una manera sencilla y factible para ingenieros que no tengan experiencia con el entorno.

En primer lugar, se analiza el diseño software de referencia que quiere ser acelerado. Posteriormente se analiza la parte del programa que tiene un alto coste computacional de recursos y se evalúan las funciones correspondientes que deben ser aceleradas a través de la interfaz hardware. Una vez comprobado, se configuran las librerías dinámicas correspondientes (para ARM 32bit) que se usarán en *SDSoC* para lograr que compile el programa. A continuación, se puede hacer clic en la opción "*Toggle HW/SW*" para convertir estas funciones a hardware y pulsando en el botón *Build*, el entorno se encarga de crear el diseño, en el que su proceso es realmente lento tardando en torno a media hora y tres cuartos de hora. Esto se debe al tiempo que emplea para la síntesis e implementación, optimización y colocación de la lógica, enrutamiento en el dispositivo y, por último, generación de flujo de bits en la FPGA (Figura 2.1.1).

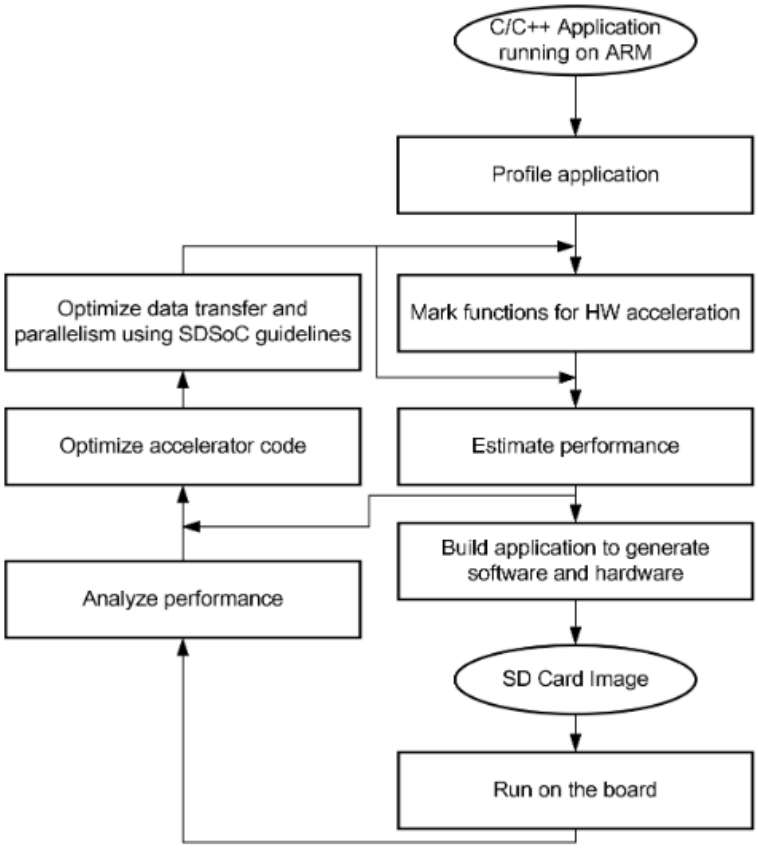


Figura 2.1.1: *SDSoC Design Flow* [16]

La etapa *Estimate performance* (Figura 2.1.1) permite mostrar al usuario una estimación del rendimiento del tiempo mejorado frente al requerido solamente en el procesador. La estimación de rendimiento asume la peor latencia de casos de las funciones de hardware y también supone el peor tamaño de transferencia de datos para las matrices. Si la latencia de la función hardware y el tamaño de la transferencia de datos en tiempo de ejecución son menores, la estimación de las prestaciones será más pesimista que el propio rendimiento real.

Finalmente, se inicia la realización del diseño total completo cuando previamente se ha obtenido el objetivo indicado, ejecutando el programa con la configuración de la plataforma de desarrollo deseada.

2.2 Plataforma de desarrollo Zybo

La plataforma escogida para el desarrollo global de este proyecto es *Zybo* (Figura 2.2.1), cuyo fabricante es Digilent [5]. *Zybo* es una plataforma de desarrollo de software digital y circuitos integrados de alto nivel, construida en torno a *Zynq-7000* (*XC7Z010-1CLG400C*). Esta placa proporciona varios recursos como memoria integrada, periféricos multimedia de audio y vídeo de entrada y salida, así como ranura USB y Ethernet. Además, existen 6 conectores disponibles para colocar cualquier diseño.

Este dispositivo está integrado por:

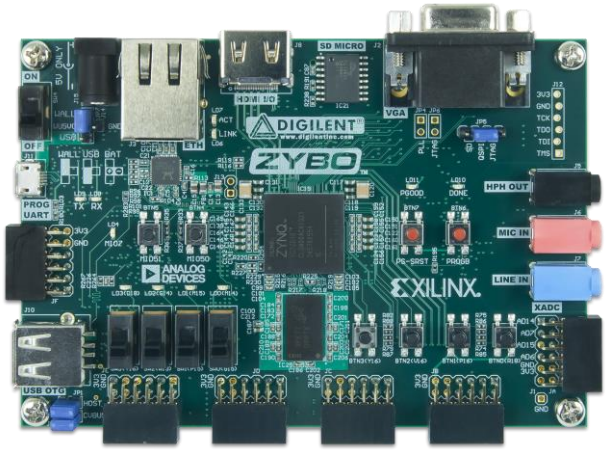


Figura 2.2.1: Plataforma *Zybo* [6]

- Memoria: 512 MB x32 DDR3 con 1050 Mbps ancho de banda
- EEPROM externa
- Memoria flash Quad-SPI de 28 Mb
- 1 conector VGA
- 1 conector HDMI
- 1 conector Ethernet de 1 GBit / 100 Mbit / 10 Mbit
- 1 puerto USB 2.0 PHY
- Ranura de tarjeta MicroSD, lugar en el que se guardan los archivos configurados
- Códec audio con salida de auriculares y micrófono
- GPIO: 6 pulsadores, 4 interruptores y 5 LEDs
- Micro-USB UART
- Micro-USB JTAG
- Micro-USB para fuente de alimentación de la placa 5V

El SoC *Zynq XC7Z010-1CLG400C* presenta un procesador *650 MHz Dual-core ARM Cortex-A9* y una FPGA Xilinx *Zynq-7000* (*XC7Z010-1CLG400C*). Sus dimensiones físicas son 88mm x 122mm. Sus recursos técnicos, datasheet y otras especificaciones se pueden encontrar en [5].

La plataforma de desarrollo *Zybo* presenta unos recursos totales limitados, con los que es posible obtener una medición del costo del área de la FPGA (Figura 2.2.2). Para ello se utilizan BRAM (bloques RAM compuestos por una memoria pequeña interna y rápida a la que se puede acceder en cada ciclo de reloj), DSP, FF y LUT.

Resources

Resource	Total
BRAM	60
DSP	80
FF	35200
LUT	17600

Figura 2.2.2: Recursos disponibles *Zybo*

2.3 Plataforma de desarrollo Zybo Z7

Posteriormente, se ha utilizado la plataforma *Zybo Z7* (Figura 2.3.1), cuyo fabricante es Digilent [9]. *Zybo Z7* reemplaza a la placa de desarrollo *Zybo* que gradualmente se irá eliminando de la producción. Los diseños son muy similares, sin embargo, *Zybo Z7* agrega varias características nuevas y mejoras de rendimiento. Para ayudar a la migración de *Zybo* a *Zybo Z7*, existe una guía de migración disponible en [10].

Este dispositivo está integrado por:

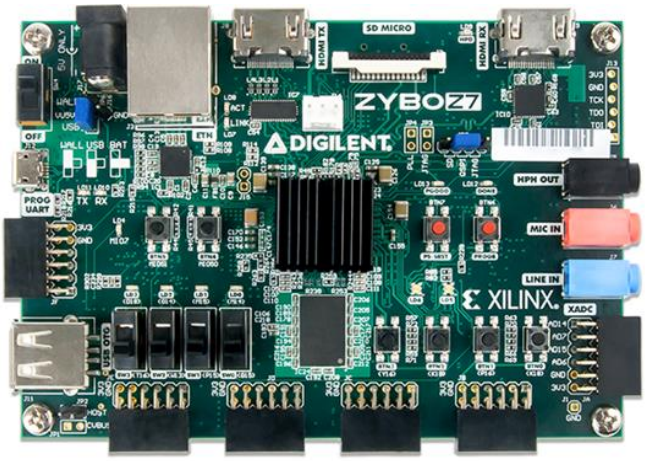


Figura 2.3.1: Plataforma *Zybo Z7* [12]

- Memoria: 1 GB x32 DDR3L
- Memoria flash Quad-SPI de 16 Mb
- 2 conectores HDMI (input/output)
- 1 conector Ethernet de 1 GBit
- 1 puerto USB 2.0 PHY
- Ranura de tarjeta MicroSD, lugar en el que se guardan los archivos configurados
- Códec audio con salida de auriculares y micrófono
- GPIO: 6 pulsadores, 4 interruptores y 5 LEDs
- Micro-USB UART
- Micro-USB JTAG
- Micro-USB para fuente de alimentación de la placa 5V

Aparte de todos los recursos que incorpora, la novedad es que incluye un conector Pcam de vídeo compatible con MIPI CSI-2, además de una entrada y salida HDMI, así como un alto ancho de banda DDR3L.

El SoC *Zynq XC7Z020-1CLG400C* presenta un procesador *667 MHz Dual-core ARM Cortex-A9* y una FPGA Xilinx *Zynq-7000* (*XC7Z020-1CLG400C*). Sus dimensiones físicas son exactamente iguales a la *Zybo*: 88mm x 122mm. Sus recursos técnicos, datasheet y otras especificaciones se pueden encontrar en [9].

Al igual que en la plataforma anterior, *Zybo Z7* presenta unos recursos totales limitados con los que se calcula una medición del costo del área de la FPGA (Figura 2.3.2). Entre ellas se encuentran BRAM, DSP, FF y LUT.

Resources

Resource	Total
BRAM	140
DSP	220
FF	106400
LUT	53200

Figura 2.3.2: Recursos totales *Zybo Z7*

Debido a que la placa de desarrollo es más reciente y moderna, los recursos son mucho mayores en comparación con el dispositivo *Zybo* anterior. Se puede comprobar (Figura 2.3.3) como *Zybo Z7* presenta un 2-3% más de recursos aproximadamente.

Resources	Zybo	Zybo Z7	Porcentaje (%)
BRAM	60	140	2,333
DSP	80	220	2,75
FF	35200	106400	3,023
LUT	17600	53200	3,023

Figura 2.3.3: Comparativa recursos placas de desarrollo *Zybo - Zybo Z7*

También es posible indicar la frecuencia de reloj con la que trabajar en la red de movimiento de datos entre la plataforma y las funciones hardware (Figura 2.3.4). En este caso, dejamos la frecuencia de reloj que viene por defecto 100.00 MHz.

Clock Frequencies

Clock	Frequency (MHz)
CPU	666.67
PL 2	150.00
PL 3	115.38
PL 4	75.00
PL 5	50.00
PL 0	100.00
PL 1	133.33

Figura 2.3.4: Frecuencias de reloj *Zybo Z7*

3 Procesamiento de imágenes

3.1 Descripción

Se llama procesamiento de imágenes al conjunto de procesos que se aplican con el objetivo de obtener una mejora significativa en la imagen que va a ser tratada, aumentando la calidad y apariencia visual de la misma, así como resaltando características importantes.

En la actualidad, existen multitud de métodos que se pueden utilizar para el tratamiento de imágenes con el fin de mejorar algún aspecto en concreto. Las operaciones más típicas suelen ser la manipulación de píxeles, con la umbralización y la binarización, el suavizado a través de filtros, compresión de la imagen, extracción de características, rotación, traslación y cambios de escala, detección de bordes, eliminación de ruido, nitidez, brillantez…

En este sentido, en el entorno de desarrollo se hará uso de la biblioteca OpenCV [14] y la librería dinámica xfOpenCV [18].

3.1.1 OpenCV

OpenCV [14] es una biblioteca de código libre que contiene más de 500 funciones útiles para el ámbito del procesamiento de imagen, incluyendo visión artificial con reconocimiento facial y de objetos. Las funciones escritas están optimizadas en C y C++ disponible tanto para equipos Windows y Mac, como para Linux. Esta librería tiene gran popularidad debido a que los desarrolladores pueden usar el código abierto y modificarlo como deseen para sus requerimientos.

Para el desarrollo de este proyecto se han empleado concretamente funciones relacionadas con el tratamiento de imágenes y extracción de características. En este sentido, se necesita que la librería funcione correctamente en procesadores ARM, es decir, aquellos en los que se basan las placas de desarrollo *Zybo* y *Zybo Z7*.

Para ello, ha sido obligatorio descargar del repositorio GitHub de OpenCV [1] la biblioteca entera y clonarla en la carpeta adecuada. La versión almacenada es la OpenCV 3.4.0 [15]. Se abre la terminal de comandos y se ejecuta:

git clone https://github.com/opencv/opencv

Una vez finalizada la descarga, hace falta compilar las librerías con la instrucción make. Algunas de las más importantes que se hacen uso son opencv_core, opencv_flann, opencv_imgcodecs, opencv_imgproc, opencv_features2d y opencv_calib3d.

3.1.2 Librería xfOpenCV

La librería xfOpenCV [18] es una biblioteca optimizada y soportada para dispositivos FPGA de Xilinx, compuesta por más de 50 kernels, basada principalmente en la biblioteca de visión por computador OpenCV. Se trata de una librería de dominio público, sencilla y con amplio soporte a procesamiento de imágenes.

En primer lugar, se debe descargar una copia de la librería del repositorio GitHub de Xilinx [3] y clonarla en un directorio local propio, ya que posteriormente se hará uso de ella. Para ello, se ejecuta el siguiente comando en la terminal:

git clone https://github.com/Xilinx/xfopencv xfopencv

Donde "xfopencv" es el nombre del directorio en el que se almacenará el repositorio en nuestro sistema local. Este es el único software requerido que debemos descargar.

La biblioteca xfOpenCV está creada para trabajar con dispositivos *Zynq* y *Zynq Ultrascale+ FPGA*. Es necesario tener instalada una versión del entorno de desarrollo *SDSoC 2017.1* en adelante para trabajar con la librería, puesto que versiones antiguas no son válidas para dichos algoritmos.

La arquitectura organizativa en la que está estructurada la librería está compuesta por:

Fichero ***examples***: Todas las funciones hardware de la biblioteca. Esta sección proporciona detalles de las funciones de procesamiento de imágenes implementadas usando una combinación de varias funciones.

Fichero ***include***: Las cabeceras necesarias para trabajar con las funciones de la librería xfOpenCV, organizadas en 4 carpetas:

- *common*: contiene las cabeceras comunes de la biblioteca.
- *core*: contiene las cabeceras de funciones matemáticas.
- *features*: contiene las cabeceras de funciones de extracción y detección de características específicas.
- *imgproc*: contiene el resto de funciones.

Para consultar más información acerca de sus modelos de uso y sus funcionalidades, se puede encontrar en [18].

3.2 Algoritmos de procesamiento

Como se ha comentado, partiendo de funcionalidades ya existentes de la librería xfOpenCV, se implementarán varios diseños de procesamiento de imágenes utilizando una combinación de varias funciones tanto en software como en hardware. El objetivo es conseguir una estimación de recursos, rendimiento y latencia de los diseños sintetizados. Estas funciones están especificadas para ser implementadas en hardware, se pueden organizar en varios grupos significativos que se detallarán más adelante (Tabla 3.2.1).

Funciones	Filtros	Cálculos	Procesamiento de entrada	Otros
	Gaussian Filter	Madd & Mult	Remap	Canny
	Sobel Filter	Accumulate	Resize	Lknpyroflow
	Bilateral Filter	Accumulate Squared	Lut	PyrDown
	Median Blur Filter	Accumulate Weighted	Channel Extract	PyrUp
	Scharr Filter	Phase		Fast
	Dilate	Magnitude		Harris
	Erode			WarpAffine
				WarpPerspective
				WarpTransform
				Threshold

Tabla 3.2.1: Funciones librería xfOpenCV

Para más información acerca de las funciones de la librería dinámica, así como su uso en hardware, se puede consultar en [18].

3.3 Imágenes

Para el desarrollo de estos algoritmos se han utilizado varias imágenes de prueba con el objetivo de conseguir diferentes resultados. Las imágenes "*im0*" y "*im1*" (Figura 3.3.1 y Figura 3.3.2) son dos imágenes similares de igual tamaño 1920x1080 píxeles. La Figura 3.3.3 ("*fox*") es una imagen de dimensiones 1280x720 píxeles obtenida del seminario de Digilent [4]. La Figura 3.3.4 ("*mountain*") corresponde a una imagen 1920x1080 píxeles, al igual que "*car*" (Figura 3.3.5). La Figura 3.3.6 corresponde a una imagen "*square*" de dimensiones 1280x1080 píxeles. Y la famosa "*lenna*", cuyo tamaño es 220x220 píxeles (Figura 3.3.7).

Las imágenes "*im0*", "*im1*", "*car*" y "*square*" vienen incluidas en el repositorio de referencia de Xilinx.

Figura 3.3.1: Imagen "*im0*" original

Figura 3.3.2: Imagen "*im1*" original

Figura 3.3.3: Imagen "*fox*" original

Figura 3.3.4: Imagen "*mountain*" original

Figura 3.3.5: Imagen "*car*" original

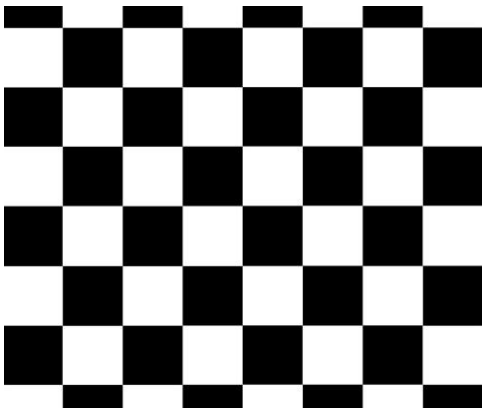


Figura 3.3.6: Imagen "*square*" original

Figura 3.3.7: Imagen "*lenna*" original

4 Procesamiento de vídeo

4.1 Descripción

El procesamiento de secuencias de vídeo en tiempo real se ha popularizado enormemente en los últimos años, gracias en gran medida al auge de las investigaciones y avances, así como la aparición de nuevos componentes que van saliendo al mercado día tras día, en el que frecuentemente se interactúa con grabaciones de vídeo. La finalidad es poder realizar una demostración que muestre una secuencia infinita de frames con una cámara de vídeo a color. El procesamiento de vídeo se ha llevado a cabo gracias a la cámara Pcam 5C desarrollada por Digilent [7].

4.1.1 Zybo Z7 Pcam C5

La plataforma *Zybo Z7* únicamente es compatible a partir de la versión de *SDSoC 2017.4* en adelante, por lo que ha sido necesario instalarse esta nueva versión avanzada. Posteriormente para poder utilizar esta plataforma, se debe descargar el repositorio de GitHub de Digilent [2] con el objetivo de configurarla en la herramienta de desarrollo y hacer uso de ella.

Pcam 5C (Figura 4.1.1) es un módulo de imágenes para su uso en placas de desarrollo FPGA, en este caso se va a emplear con la plataforma *Zybo Z7*. El módulo está formado por un sensor de imagen en color OmniVision OV5640 de 5 megapíxeles (MP), cuyo sensor presenta diversas funcionalidades de procesamiento interno que hablaremos más adelante.

Los datos son transferidos a través de una interfaz MIPI CSI-2 que proporciona un ancho de banda lo suficientemente eficiente y extenso para adaptar la transmisión de vídeo a 1080p (30 fotogramas por segundo) y 720p (60 fotogramas por segundo).

Este módulo es compatible con la plataforma *Zybo Z7* mediante un cable flexible de 10 cm con 15 pines y una lente M12 con enfoque fijo. También se puede conectar en otros dispositivos, como la famosa placa Raspberry Pi [7].

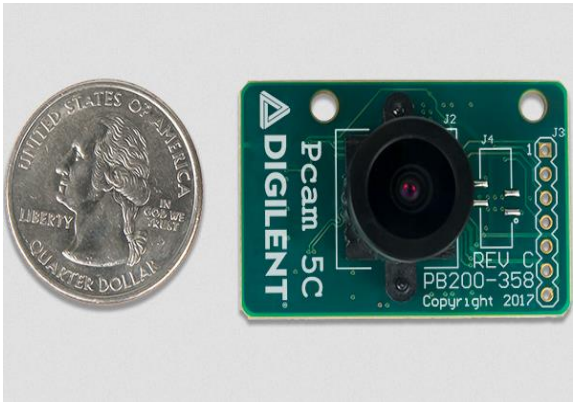


Figura 4.1.1: Módulo *Pcam 5C* [8]

Este módulo presenta las siguientes características:

- Sensor de imagen en color de 5 megapíxeles (MP)
- Interfaz de sensor de imagen MIPI CSI-2
- Formato de salida: RAW10, RGB565, CCIR656, YUV422/420, YCbCr422 y compresión JPEG
- Montura lente estándar M12
- Cable de 10 cm y una lente de enfoque fijo
- Compatible con placas de desarrollo accesibles con Pcam Digilent
- Dimensiones físicas 25 mm x 40 mm

4.2 Demostración de procesamiento

A partir del proyecto de código abierto Zybo Z7 Pcam 5C [13], se decidió adquirir originalmente la última revisión del módulo Pcam 5C como fuente de vídeo para reenviar los datos de transmisión de imágenes al puerto HDMI TX de la plataforma *Zybo Z7* y mostrarlo por pantalla.

El comportamiento consiste en transmitir los datos de vídeo a través del puerto Pcam finalizando en el puerto HDMI TX del dispositivo. Para ello, se usa el canal de comunicación UART (Figura 4.2.1) a través de una terminal de comandos, conectándolo al dispositivo tty a 115200 baudios, sin control de flujo y sin paridad (*screen /dev/ttyUSB1 115200*). De esta manera, se dispone del control del sensor de imagen y un menú de procesamiento posterior (Figura 4.2.1).

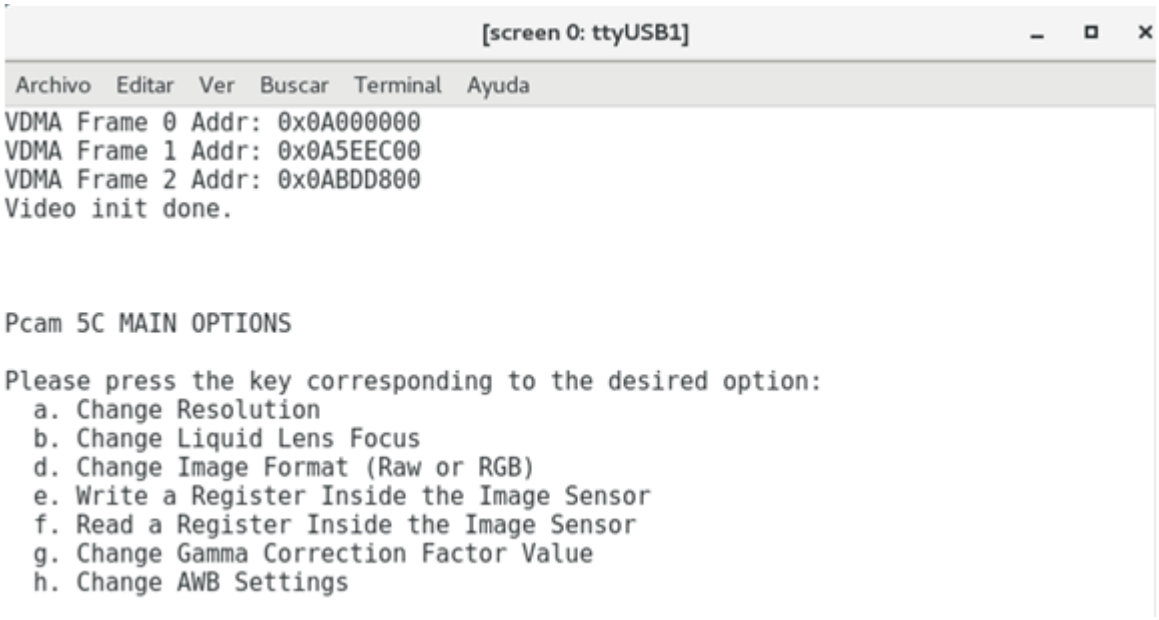


Figura 4.2.1: Canal de comunicación UART

Este menú desplegable y dinámico presenta diferentes opciones que el usuario puede escoger para por ejemplo mejorar la calidad de la imagen, cambiar el balance de blancos automático, la calibración automática de nivel de negro, controles para ajustar la saturación, el tono, la gamma y la nitidez...

- ***Opción A***: Resolución. Esta opción permite modificar la resolución del vídeo procedente del sensor. Existen tres modos utilizables:

    1. 1280 x 720, 60fps
    2. 1920 x 1080, 15fps
    3. 1920 x 1080, 30fps

- ***Opción B***: Foco de la lente líquida. Esta opción no es compatible con esta versión de Pcam 5C y por lo tanto debe ser ignorada.

    Please enter value of liquid lens register, in hex, with small letters: 0x

- ***Opción D***: Formato de imagen. Esta opción fue creada expresamente para temas de depuración y siempre debe configurarse la opción 2 (RAW).

    1. Select image format to be RGB, output still Raw
    2. Select image format & output to both be Raw

- ***Opción E***: Escribir un registro dentro del sensor de imagen. Esta opción permite escribir un valor en cualquier registro dentro del sensor de imagen sobre la interfaz OmniVision SCCB. Esta opción es muy útil para explorar las características específicas del sensor de imagen.

    Please enter address of image sensor register, in hex, with small letters:

- ***Opción F***: Leer un registro dentro del sensor de imagen. Esta opción permite leer un valor de cualquier registro dentro del sensor de imagen sobre la interfaz OmniVision SCCB.

    Please enter address of image sensor register, in hex, with small letters:

- ***Opción G***: Factor de corrección gamma. Esta opción permite cambiar el valor del factor de corrección de gamma que se realiza mediante la IP de transmisión AXI dentro de la FPGA. Existen 5 valores del factor de corrección gama:

    1. Gamma Factor = 1
    2. Gamma Factor = 1/1.2
    3. Gamma Factor = 1/1.5
    4. Gamma Factor = 1/1.8
    5. Gamma Factor = 1/2.2

- ***Opción H***: Configuración AWB. Esta opción permite ajustar la forma en el que el sensor de imagen realiza el balance automático de blancos. Existen 3 modos utilizables: avanzado, simple y deshabilitado.

    1. Enable Advanced AWB
    2. Enable Simple AWB
    3. Disable AWB

Para más información con respecto a las opciones de procesamiento, se puede consultar en [11].

Es cierto que se ha conseguido programar satisfactoriamente el dispositivo con la placa de desarrollo, así como la generación del bitstream con éxito y el menú desplegable. Sin embargo, no ha sido posible que la cámara de vídeo funcionase a pleno rendimiento debido a que no muestra correctamente lo que se ha configurado.

5 Integración, pruebas y resultados

En esta sección, se va a llevar a cabo un análisis de los resultados obtenidos en el procesamiento de imágenes en la placa de desarrollo *Zybo* que se ha ido tomando tanto en versión software como en la integración en hardware que es el objetivo final del proyecto. En este sentido, por una parte, se han ido anotando diferentes mediciones y estimaciones de latencia y rendimiento en cuanto a tiempos de ejecución y, por otro lado, la cantidad de recursos empleados y área utilizada por cada uno de los módulos implementados respecto del total. Es importante tener en cuenta que todos ellos se procesarán en escala de grises, salvo el algoritmo fast.

Las funciones analizadas para el procesamiento de imágenes se pueden clasificar en varios grupos: filtros, cálculos, procesamiento de entrada y otros. Finalmente, se hará una comparativa de todos los subgrupos para observar de manera más visual todos los resultados examinados.

La estimación de aceleración que hace *SDSoC* se calcula comparando un mismo diseño software con la versión hardware. De este modo, se puede evaluar cuánto se ha acelerado la función.

5.1 Filtros

Se han evaluado distintas funciones de filtros que modifican el estado de la imagen original. En general, los filtros utilizan técnicas que hacen que la imagen resultante sea más suave, es decir, se aplica una reducción en la variación de intensidad entre píxeles cercanos. Además, es deseable disminuir el ruido que puede existir, en el que el nivel de intensidad entre un píxel y sus vecinos más próximos son muy diferentes y se resaltan notablemente.

Otro aspecto a tener en cuenta es la detección y realce de bordes que origina un cambio repentino en la intensidad de los píxeles. Por eso mismo, los filtros pretenden conseguir un efecto particular en la imagen procesada.

5.1.1 Gaussian Filter

El filtro Gaussiano aplica un desenfoque gaussiano en la imagen de entrada. El filtrado gaussiano se realiza al convolucionar cada punto de la imagen de entrada con un núcleo gaussiano. Estos filtros son muy utilizados para la detección de bordes y buenos para eliminar el ruido gaussiano.

$$G_0(x,y) = e^{\frac{-(x-\mu_x)^2}{2\sigma_x{}^2}+\frac{-(y-\mu_y)^2}{2\sigma_y{}^2}}$$

Ecuación 5.1: Función gaussiana [18]

μ_x, μ_y son los valores medios y σ_x, σ_y son las varianzas en las direcciones x e y respectivamente. En este caso, los valores de μ_x, μ_y se consideran como ceros y los valores de σ_x, σ_y son iguales.

Image	Performance estimates for 'gaussian filter' function	
	Estimated hardware latency	15702777
im0	SW-only (Measured cycles)	465677062
	Estimated speedup	29,656

Figura 5.1.1: Rendimiento estimado algoritmo Gaussian Filter

Resource utilization estimates for hardware accelerators			
RESOURCE	USED	TOTAL	UTILIZATION (%)
DSP	16	80	20
BRAM	10	60	16,667
LUT	11096	17600	63,045
FF	10031	35200	28,497
HW accelerated (cycles)			Speedup
im0	13622568		34,184

Figura 5.1.2: Recursos aceleración hardware Gaussian Filter

5.1.2 Sobel Filter

El filtro Sobel calcula los gradientes de la imagen de entrada en ambas direcciones, x e y, al convolucionar un tamaño de kernel con la imagen de entrada que se procesa. El kernel puede ser de dimensiones 3x3, 5x5 o 7x7, en este caso el tamaño de la ventana usado es 3x3. Este filtro es poco sensible al ruido y obtiene una buena respuesta en bordes verticales y horizontales.

Image	Performance estimates for 'sobel filter' function	
	Estimated hardware latency	47187447
im0	SW-only (Measured cycles)	665125204
	Estimated speedup	14,095
lenna	SW-only (Measured cycles)	15738334
	Estimated speedup	0,334
mountain	SW-only (Measured cycles)	662902016
	Estimated speedup	14,048
fox	SW-only (Measured cycles)	295344168
	Estimated speedup	6,259

Figura 5.1.3: Rendimiento estimado algoritmo Sobel Filter

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	0	80	0
BRAM	17	60	28,333
LUT	10250	17600	58,239
FF	12048	35200	34,227
HW accelerated (cycles)			**Speedup**
im0	13592200		48,934
lenna	337752		46,597
mountain	13592016		48,771
fox	6081704		48,563

Figura 5.1.4: Recursos aceleración hardware Sobel Filter

5.1.3 Bilateral Filter

El filtro Bilateral se puede utilizar para conservar los bordes mientras se suaviza la imagen. De forma análoga al filtro gaussiano, el filtro bilateral también considera los píxeles vecinos con pesos asignados a cada uno de ellos. Estos pesos tienen dos componentes, el primero es el mismo usado por el filtro gaussiano. El segundo componente toma en cuenta la diferencia en la intensidad entre los píxeles vecinos cercanos y el evaluado.

Image	Performance estimates for 'bilateral filter' function	
	Estimated hardware latency	205931143
im0	SW-only (Measured cycles)	1655559310
	Estimated speedup	8,039
mountain	SW-only (Measured cycles)	1664942406
	Estimated speedup	8,085

Figura 5.1.5: Rendimiento estimado algoritmo Bilateral Filter

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	19	80	23,75
BRAM	11	60	18,333
LUT	13773	17600	78,256
FF	12120	35200	34,432
HW accelerated (cycles)			**Speedup**
im0	138651128		11,940
mountain	138612781		12,011

Figura 5.1.6: Recursos aceleración hardware Bilateral Filter

5.1.4 Median Blur Filter

El filtro Median Blur realiza una operación de filtro mediana en la imagen de entrada. El filtro mediano actúa como un filtro digital no lineal que mejora la reducción de ruido. El

tamaño de la ventana del filtro puede cambiarse por cualquier número entero positivo impar mayor que 1, luego se ha realizado para tamaños 1 y 3.

Image	Performance estimates for 'median blur filter' function	
	Estimated hardware latency	20359076
im0	SW-only (Measured cycles)	465836644
	Estimated speedup	22,881

Figura 5.1.7: Rendimiento estimado algoritmo Median Blur Filter

Resource utilization estimates for hardware accelerators			
RESOURCE	USED	TOTAL	UTILIZATION (%)
DSP	0	80	0
BRAM	9	60	15
LUT	6857	17600	38,960
FF	7170	35200	20,369
HW accelerated (cycles)			Speedup
im0	13629164		34,179

Figura 5.1.8: Recursos aceleración hardware Median Blur Filter

5.1.5 Scharr Filter

El filtro Scharr calcula los gradientes de la imagen de entrada en ambas direcciones, x e y, convolucionando el kernel con la imagen de entrada que se procesa.

Image	Performance estimates for 'scharr filter' function	
	Estimated hardware latency	47187477
im0	SW-only (Measured cycles)	729114284
	Estimated speedup	15,451
mountain	SW-only (Measured cycles)	735051192
	Estimated speedup	15,577

Figura 5.1.9: Rendimiento estimado algoritmo Scharr Filter

Resource utilization estimates for hardware accelerators			
RESOURCE	USED	TOTAL	UTILIZATION (%)
DSP	0	80	0
BRAM	17	60	28,333
LUT	10582	17600	60,125
FF	12253	35200	34,810
HW accelerated (cycles)			Speedup
im0	13611466		53,566
mountain	13609190		54,011

Figura 5.1.10: Recursos aceleración hardware Scharr Filter

5.1.6 Dilate

La función Dilate reemplaza la intensidad del píxel actual por el valor máximo de la intensidad de un bloque vecino de tamaño 3x3.

Image	Performance estimates for 'dilate' function	
	Estimated hardware latency	20309046
im0	SW-only (Measured cycles)	418797496
	Estimated speedup	20,621

Figura 5.1.11: Rendimiento estimado algoritmo Dilate

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	0	80	0
BRAM	9	60	15
LUT	6000	17600	34,091
FF	6726	35200	19,108
HW accelerated (cycles)			**Speedup**
im0	13606152		30,780

Figura 5.1.12: Recursos aceleración hardware Dilate

5.1.7 Erode

La función Erode reemplaza la intensidad del píxel actual por el valor mínimo de la intensidad de un bloque vecino de tamaño 3x3.

Image	Performance estimates for 'erode' function	
	Estimated hardware latency	20309046
im0	SW-only (Measured cycles)	417658402
	Estimated speedup	20,565
mountain	SW-only (Measured cycles)	417678278
	Estimated speedup	20,566

Figura 5.1.13: Rendimiento estimado algoritmo Erode

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	0	80	0
BRAM	9	60	15
LUT	6000	17600	34,091
FF	6726	35200	19,108
HW accelerated (cycles)			**Speedup**
im0	13625640		30,652
mountain	13615016		30,678

Figura 5.1.14: Recursos aceleración hardware Erode

5.2 Cálculos

Se han evaluado diferentes funciones con cálculos computacionales que tratan la imagen original. Este grupo comprende operaciones sencillas de procesamiento como es la suma de imágenes.

5.2.1 Madd & Mult

La implementación de una multiplicación y una suma de una matriz de tamaño 32x32 usando valores de 4-byte de punto flotante. Por defecto, ambas funciones están especificadas para implementarlas en hardware, aunque se pueden ejecutar también en software para verificar el resultado.

Performance estimates for 'madd & mult' function			
1024 iterations	Estimated hardware latency		20813
	SOFTWARE	Average CPU cycles in software	218693
		Average CPU cycles in hardware	227822
		Speed up	0.959929
	HARDWARE	Average CPU cycles in software	304458
		Average CPU cycles in hardware	19764
		Speed up	15,4047

Figura 5.2.1: Rendimiento estimado algoritmo Madd & Mult

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	32	80	40
BRAM	48	60	80
LUT	17265	17600	98,097
FF	23951	35200	68,043
HW accelerated (cycles)			**Speedup**
1024 iterations	19764		15,4047

Figura 5.2.2: Recursos aceleración hardware Madd & Mult

Estas funciones son las primeras que se han realizado gracias al tutorial existente que ha servido de gran ayuda para coger práctica con la plataforma de desarrollo [17].

5.2.2 Accumulate

La función Accumulate realiza la suma una imagen (src1) a la imagen del acumulador (src2) y genera una imagen resultado acumulada (dst).

$$dst(x, y) = src1(x, y) + src2(x, y)$$

Ecuación 5.2: Función acumulada [18]

Image	Performance estimates for 'accumulate' function	
im0 + mountain	Estimated hardware latency	19302192
	SW-only (Measured cycles)	580631408
	Estimated speedup	30,081

Figura 5.2.3: Rendimiento estimado algoritmo Accumulate

Resource utilization estimates for hardware accelerators			
RESOURCE	USED	TOTAL	UTILIZATION (%)
DSP	0	80	0
BRAM	16	60	26,667
LUT	9279	17600	52,722
FF	11532	35200	32,761
HW accelerated (cycles)			Speedup
im0 + mountain	13595530		42,708

Figura 5.2.4: Recursos aceleración hardware Accumulate

5.2.3 Accumulate Squared

La función Accumulate Squared realiza la suma del cuadrado de una imagen $(src1)^2$ con otra imagen (src2) y genera una imagen resultado acumulada (dst).

$$dst(x, y) = src1(x, y)^2 + src2(x, y)$$

Ecuación 5.3: Función acumulativa cuadrada [18]

Image	Performance estimates for 'accumulate squared' function	
	Estimated hardware latency	19296738
im0 + im1	SW-only (Measured cycles)	585218986
	Estimated speedup	30,327
im0 + mountain	SW-only (Measured cycles)	584681136
	Estimated speedup	30,299

Figura 5.2.5: Rendimiento estimado algoritmo Accumulate Squared

Resource utilization estimates for hardware accelerators			
RESOURCE	USED	TOTAL	UTILIZATION (%)
DSP	1	80	1,25
BRAM	16	60	26,667
LUT	9342	17600	53,080
FF	11555	35200	32,827
HW accelerated (cycles)			Speedup
im0 + im1	13624752		42,953
im0 + mountain	13604452		42,977

Figura 5.2.6: Recursos aceleración hardware Accumulate Squared

5.2.4 Accumulate Weighted

La función Accumulate Weighted calcula la suma ponderada de una imagen de entrada (src1) con otra imagen (src2) y genera una imagen resultado (dst). El factor alpha es el peso aplicado a la imagen de entrada cuyos valores se encuentran entre 0 y 1. En este caso, se ha escogido un alpha igual a 0.76.

$$dst(x, y) = alpha*src1(x, y) + (1 - alpha)*src2(x, y)$$

Ecuación 5.4: Función acumulativa ponderada [18]

Image	Performance estimates for 'accumulate weighted' function	
	Estimated hardware latency	19304024
im0 + im1	SW-only (Measured cycles)	590850004
	Estimated speedup	30,608
im0 + mountain	SW-only (Measured cycles)	593431032
	Estimated speedup	30,741

Figura 5.2.7: Rendimiento estimado algoritmo Accumulate Weighted

Resource utilization estimates for hardware accelerators			
RESOURCE	USED	TOTAL	UTILIZATION (%)
DSP	5	80	6,25
BRAM	16	60	26,667
LUT	10142	17600	57,625
FF	12201	35200	34,662
HW accelerated (cycles)			Speedup
im0 + im1	13609848		43,413
im0 + mountain	13613066		43,593

Figura 5.2.8: Recursos aceleración hardware Accumulate Weighted

5.2.5 Phase

La función Phase calcula los ángulos polares de dos imágenes. Las imágenes de entrada son imágenes x-gradient e y-gradient de 16 bits. La imagen de salida es del mismo tipo que la imagen de entrada. Se puede definir dos formatos de salida, en radianes o grados.

For radians:

$$angle(x, y) = atan2(g_y, g_x)$$

For degrees:

$$angle(x, y) = atan2(g_y, g_x)*\frac{180}{\pi}$$

Ecuación 5.5: Función fase polar [18]

Image	Performance estimates for 'phase' function	
	Estimated hardware latency	53926677
im0	SW-only (Measured cycles)	778468020
	Estimated speedup	14,436
mountain	SW-only (Measured cycles)	784680716
	Estimated speedup	14,551

Figura 5.2.9: Rendimiento estimado algoritmo Phase

Resource utilization estimates for hardware accelerators			
RESOURCE	USED	TOTAL	UTILIZATION (%)
DSP	4	80	5
BRAM	19	60	31,667
LUT	11122	17600	63,193
FF	13329	35200	37,866
HW accelerated (cycles)			Speedup
im0	13675786		56,923
mountain	13666298		57,417

Figura 5.2.10: Recursos aceleración hardware Phase

5.2.6 Magnitude

La función Magnitude calcula la magnitud de las imágenes. Las imágenes de entrada son imágenes x-gradient y y-gradient de 16 bits. La imagen de salida es del mismo tipo que la imagen de entrada. Para la normalización L1NORM, la imagen calculada es la suma de los valores absolutos de x-gradient e y-gradient, mientras que para la normalización L2NORM, la imagen corresponde a la raíz cuadrada de la suma de los cuadrados de x-gradient e y-gradient.

$$g = |g_x| + |g_y| \qquad g = \sqrt{(g_x^2 + g_y^2)}$$

Ecuación 5.6: Función magnitud L1NORM y L2NORM respectivamente [18]

Image	Performance estimates for 'magnitude' function	
	Estimated hardware latency	53926677
im0	SW-only (Measured cycles)	624973614
	Estimated speedup	11,589
mountain	SW-only (Measured cycles)	627298350
	Estimated speedup	11,632

Figura 5.2.11: Rendimiento estimado algoritmo Magnitude

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	0	80	0
BRAM	16	60	26,667
LUT	9657	17600	54,869
FF	11631	35200	33,043
HW accelerated (cycles)			**Speedup**
im0	13593486		45,976
mountain	13589278		46,161

Figura 5.2.12: Recursos aceleración hardware Magnitude

5.3 Procesamiento de entrada

Se han evaluado distintas funciones que emplean procesamiento de entrada para modificar la imagen original. En este apartado se describen aquellas transformaciones geométricas que se llevan a cabo con interpolaciones de la imagen inicial.

La interpolación aplicada por el píxel vecino más cercano (*nearest neighbor* en inglés) es un método básico consistente en asignar el valor del píxel que se va a interpolar al píxel más contiguo. Este procedimiento simplemente incrementa el tamaño de cada píxel requiriendo un tiempo de procesado bajo. Sin embargo, la interpolación bilineal comprueba los valores de un bloque de píxeles determinado, en el que se realiza el promedio de todos ellos y el resultado es el valor que se va a interpolar. Por el contrario, este sistema tomará más tiempo de procesamiento.

5.3.1 Remap

La función Remap toma píxeles de un lugar concreto de la imagen y los coloca en otra posición. Para mapear la imagen desde la imagen fuente hasta la imagen de destino se pueden usan dos tipos de métodos de interpolación. El primero de ellos es la interpolación por el píxel vecino más próximo y el otro es interpolación bilineal.

Image	Performance estimates for 'remap' function	
	Estimated hardware latency	80888078
im0	SW-only (Measured cycles)	1273452612
	Estimated speedup	15,743
mountain	SW-only (Measured cycles)	1273311194
	Estimated speedup	15,742

Figura 5.3.1: Rendimiento estimado algoritmo Remap

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	13	80	16,25
BRAM	36	60	60
LUT	18095	17600	102,813
FF	20189	35200	57,355
HW accelerated (cycles)			**Speedup**
im0	-		-
mountain	-		-

Figura 5.3.2: Recursos aceleración hardware Remap

No es posible llevarla a hardware debido que hace un uso excesivo de recursos LUT.

5.3.2 Resize

La función Resize se utiliza para cambiar el tamaño de la imagen original al tamaño que queramos en una imagen de destino. Se pueden usar diferentes tipos de técnicas de interpolación para modificar el tamaño, por ejemplo, la interpolación del píxel vecino más cercano, bilineal y de área. El tipo de interpolación se puede indicar como un parámetro de entrada.

Image	Performance estimates for 'resize' function	
im0	Estimated hardware latency	20573719
AREA	SW-only (Measured cycles)	529603912
	Estimated speedup	25,742
BILINEAL	SW-only (Measured cycles)	396785344
	Estimated speedup	19,286
VECINO CERCANO	SW-only (Measured cycles)	286320278
	Estimated speedup	13,917

Figura 5.3.3: Rendimiento estimado algoritmo Resize

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	48	80	60
BRAM	23	60	38,333
LUT	16509	17600	93,801
FF	30415	35200	86,406
HW accelerated (cycles)			**Speedup**
AREA	13846442		38,248
BILINEAL	13603810		29,167
VECINO CERCANO	13572156		21,096

Figura 5.3.4: Recursos aceleración hardware Resize

Las dimensiones de las imágenes se han visto reducidas a 640x480 píxeles.

5.3.3 LUT (Look Up Table)

La función LUT realiza la operación de búsqueda en tablas. Transforma la imagen de origen en una imagen de destino usando la tabla de búsqueda dada.

Image	Performance estimates for 'lut' function	
	Estimated hardware latency	20440085
im0	SW-only (Measured cycles)	320381174
	Estimated speedup	15,674

Figura 5.3.5: Rendimiento estimado algoritmo LUT

Resource utilization estimates for hardware accelerators			
RESOURCE	USED	TOTAL	UTILIZATION (%)
DSP	0	80	0
BRAM	16	60	26,667
LUT	9858	17600	56,011
FF	11642	35200	33,074
HW accelerated (cycles)			Speedup
im0	9076852		35,297

Figura 5.3.6: Recursos aceleración hardware LUT

5.3.4 Channel Extract

La función Channel Extract divide una matriz multicanal (datos intercalados en píxeles de 32 bits) en varias matrices de un solo canal y devuelve uno solo. El canal que se extraerá se especifica utilizando el argumento del canal y su valor se especifica con la macro definida en el tipo de datos "xf_channel_extract_e". Por ejemplo, RED, GREEN, BLUE, ALPHA y LUMA.

	Image	Performance estimates for 'channel extract' function	
		Estimated hardware latency	23088090
RED	im0	SW-only (Measured cycles)	345805256
		Estimated speedup	14,978
	mountain	SW-only (Measured cycles)	342997622
		Estimated speedup	14,856
BLUE	im0	SW-only (Measured cycles)	345845862
		Estimated speedup	14,979

Figura 5.3.7: Rendimiento estimado algoritmo Channel Extract

	Resource utilization estimates for hardware accelerators			
	RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
	DSP	6	80	7,5
	BRAM	8	60	13,333
	LUT	5735	17600	32,585
	FF	6241	35200	17,730
	HW accelerated (cycles)			**Speedup**
RED	**im0**	13606002		25,416
	mountain	13591864		25,236

Figura 5.3.8: Recursos aceleración hardware Channel Extract

5.4 Otros

El resto de funciones están clasificadas en este grupo.

5.4.1 Canny

El detector de bordes Canny encuentra los bordes de una imagen o marco de vídeo. Es uno de los algoritmos más populares para la detección de bordes que logran dotar a las imágenes de mejor nitidez. El algoritmo de Canny tiene como objetivo satisfacer tres criterios principales:

1. Baja tasa de errores: una buena detección de los bordes existentes.
2. Buena localización: la distancia entre los píxeles de borde detectados y los reales debe minimizarse.
3. Respuesta mínima: solo una respuesta del detector por borde.

Image	Performance estimates for 'canny' function	
	Estimated hardware latency	2912912315
im0	SW-only (Measured cycles)	58961982172
	Estimated speedup	20,242
lenna	SW-only (Measured cycles)	6376576980
	Estimated speedup	2,189
mountain	SW-only (Measured cycles)	59511376040
	Estimated speedup	20,430
fox	SW-only (Measured cycles)	38555499138
	Estimated speedup	13,236
square	SW-only (Measured cycles)	37867308218
	Estimated speedup	13,000

Figura 5.4.1: Rendimiento estimado algoritmo Canny

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	1	80	1,25
BRAM	15	60	25
LUT	11245	17600	63,892
FF	9345	35200	26,548
HW accelerated (cycles)			**Speedup**
im0	39570285338		1,490
lenna	4350591802		1,466
mountain	39723061884		1,498
fox	26116328496		1,476
square	25512822598		1,484

Figura 5.4.2: Recursos aceleración hardware Canny

5.4.2 Lknpyroflow

El flujo óptico es el patrón de movimiento aparente de los objetos de una imagen entre dos secuencias consecutivas, causado por el movimiento del objeto o la cámara. Es un campo vectorial 2D, donde cada vector es un vector de desplazamiento que muestra el movimiento de los puntos desde el primer frame hasta el segundo.

La función Lknpyroflow supone que las intensidades de los píxeles de un objeto no tienen demasiadas variaciones entre secuencias consecutivas y, por lo tanto, se considera que los píxeles vecinos tienen un movimiento similar.

Image	Performance estimates for 'lknpyroflow' function	
	Estimated hardware latency	377412637
car	SW-only (Measured cycles)	4702880794
	Estimated speedup	12,461

Figura 5.4.3: Rendimiento estimado algoritmo Lknpyroflow

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	35	80	43,75
BRAM	106	60	176,667
LUT	24645	17600	140,028
FF	21675	35200	61,577
HW accelerated (cycles)			**Speedup**
car	-		-

Figura 5.4.4: Recursos aceleración hardware Lknpyroflow

No es posible llevarla a hardware debido que hace un uso excesivo de recursos BRAM y LUT.

5.4.3 PyrDown

La función PyrDown es un algoritmo que suaviza una imagen antes de realizar una reducción de la misma. La imagen se suaviza utilizando un filtro gaussiano, mientras que la reducción a escala se realiza al eliminar píxeles de las filas y columnas pares.

Image	Performance estimates for 'pyrDown' function	
	Estimated hardware latency	20396646
im0	SW-only (Measured cycles)	646634482
	Estimated speedup	31,703
mountain	SW-only (Measured cycles)	646395270
	Estimated speedup	31,691

Figura 5.4.5: Rendimiento estimado algoritmo PyrDown

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	1	80	1,25
BRAM	10	60	16,667
LUT	17734	17600	100,761
FF	8394	35200	23,847
HW accelerated (cycles)			**Speedup**
im0	-		-
mountain	-		-

Figura 5.4.6: Recursos aceleración hardware PyrDown

No es posible llevarla a hardware debido que hace un uso excesivo de recursos LUT.

5.4.4 PyrUp

La función PyrUp es un algoritmo de up-sampling, es decir, modifica la frecuencia de muestreo original de las imágenes. Primero se encarga de insertar cero filas y cero columnas después de cada una hasta el tamaño de la imagen de salida. El tamaño de la imagen de salida es siempre 2*filas x 2*columnas. La imagen se suaviza posteriormente con el filtro gaussiano. Sin embargo, para compensar la intensidad de píxeles que se reduce debido al relleno de ceros, cada píxel de salida se multiplica por 4.

Image	Performance estimates for 'pyrUp' function	
	Estimated hardware latency	81219492
im0	SW-only (Measured cycles)	2425169290
	Estimated speedup	29,859
mountain	SW-only (Measured cycles)	2443557208
	Estimated speedup	30,086

Figura 5.4.7: Rendimiento estimado algoritmo PyrUp

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	0	80	0
BRAM	13	60	21,667
LUT	17692	17600	100,523
FF	8377	35200	23,798
HW accelerated (cycles)			**Speedup**
im0	-		-
mountain	-		-

Figura 5.4.8: Recursos aceleración hardware PyrUp

No es posible llevarla a hardware debido que hace un uso excesivo de recursos LUT.

5.4.5 Fast

La función Fast es un algoritmo de detección de esquinas, más rápido que la mayoría de otros detectores, encargado de evaluar cada píxel de una imagen y comparar su intensidad con los 16 píxeles vecinos en un círculo, llamado círculo de Bresenham. Si se encuentra que la intensidad de 9 píxeles contiguos es mayor o menor que la del píxel que estamos evaluando, entonces el píxel se declara como una esquina.

Image	Performance estimates for 'fast' function	
	Estimated hardware latency	20450330
im0	SW-only (Measured cycles)	287881445
	Estimated speedup	14,077
mountain	SW-only (Measured cycles)	293683227
	Estimated speedup	14,361
square	SW-only (Measured cycles)	187321672
	Estimated speedup	9,160

Figura 5.4.9: Rendimiento estimado algoritmo Fast

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	0	80	0
BRAM	21	60	35
LUT	16986	17600	96,511
FF	19213	35200	54,582
HW accelerated (cycles)			**Speedup**
im0	13724064		20,976
mountain	13724360		21,399
square	9201230		20,358

Figura 5.4.10: Recursos aceleración hardware Fast

5.4.6 Harris

La función Harris realiza un barrido píxel a píxel de una imagen con desplazamientos en la dirección "x" e "y" calculando su variación de intensidad. En caso de que encuentre una esquina con una gran variación de intensidad la redondeará.

Image	Performance estimates for 'harris' function	
	Estimated hardware latency	20324886
im0	SW-only (Measured cycles)	363894566
	Estimated speedup	17,904
mountain	SW-only (Measured cycles)	364404250
	Estimated speedup	17,929
square	SW-only (Measured cycles)	241361487
	Estimated speedup	11,875

Figura 5.4.11: Rendimiento estimado algoritmo Harris

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	12	80	15
BRAM	32	60	53,333
LUT	17453	17600	99,165
FF	16774	35200	47,653
HW accelerated (cycles)			**Speedup**
im0	13638464		26,681
mountain	13640238		26,715
square	9131786		26,431

Figura 5.4.12: Recursos aceleración hardware Harris

5.4.7 WarpAffine

La función WarpAffine se usa para operaciones de rotación, escalado y traducción de imágenes. La función toma la transformación inversa como entrada y la transforma mediante la multiplicación con una matriz de tamaño 2x3.

Image	Performance estimates for 'warpAffine' function	
	Estimated hardware latency	208560731
im0	SW-only (Measured cycles)	971645272
	Estimated speedup	4,659
mountain	SW-only (Measured cycles)	971450682
	Estimated speedup	4,658

Figura 5.4.13: Rendimiento estimado algoritmo WarpAffine

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	341	80	426,25
BRAM	110	60	183,333
LUT	33083	17600	187,972
FF	30519	35200	86,702
HW accelerated (cycles)			**Speedup**
im0	-		-
mountain	-		-

Figura 5.4.14: Recursos aceleración hardware WarpAffine

No es posible llevarla a hardware debido que hace un uso excesivo de recursos DSP, BRAM y LUT.

5.4.8 WarpPerspective

La función WarpPerspective está diseñada para realizar una transformación de la perspectiva de una imagen. Toma la transformación inversa como entrada y la transforma mediante la multiplicación con una matriz de tamaño 3x3.

Image	Performance estimates for 'warpPerspective' function	
	Estimated hardware latency	2022071062
im0	SW-only (Measured cycles)	16775285184
	Estimated speedup	8,296
mountain	SW-only (Measured cycles)	16892912026
	Estimated speedup	8,354

Figura 5.4.15: Rendimiento estimado algoritmo WarpPerspective

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	527	80	658,75
BRAM	129	60	215
LUT	39604	17600	225,023
FF	39099	35200	111,077
HW accelerated (cycles)			**Speedup**
im0	-		-
mountain	-		-

Figura 5.4.16: Recursos aceleración hardware WarpPerspective

No es posible llevarla a hardware debido que hace un uso excesivo de recursos DSP, BRAM, LUT y FF.

5.4.9 WarpTransform

La función WarpTransform realiza transformaciones geométricas y de perspectiva de una imagen. El tipo de transformación se le pasa por parámetro, se establece '0' para transformación afín y '1' para transformación de perspectiva.

Image	Performance estimates for 'warpTransform' function	
	Estimated hardware latency	206505750
im0	SW-only (Measured cycles)	1277191930
	Estimated speedup	6,185

Figura 5.4.17: Rendimiento estimado algoritmo WarpTransform

Resource utilization estimates for hardware accelerators			
RESOURCE	**USED**	**TOTAL**	**UTILIZATION (%)**
DSP	38	80	47,5
BRAM	68	60	113,333
LUT	25123	17600	142,744
FF	19737	35200	56,071
HW accelerated (cycles)			**Speedup**
im0	-		-

Figura 5.4.18: Recursos aceleración hardware WarpTransform

No es posible llevarla a hardware debido que hace un uso excesivo de recursos BRAM y LUT.

5.4.10 Threshold

La función Threshold es un método de segmentación orientado a los píxeles de una imagen de entrada. Mediante una umbralización o binarización se va comparando píxel a píxel comprobando si está por encima o por debajo de un cierto valor de umbral. Los umbrales disponibles son de dos tipos: binario y rango. En el umbral binario, se establece un valor de umbral y el píxel de salida obtendrá 0 (blanco) o 255 (negro) dependiendo si está por encima o por debajo de ese cierto umbral. Por otro lado, en el umbral de rango, se establece el valor de umbral superior e inferior y dependiendo de esos valores, la salida se establece en 0 o 255.

	Image	Performance estimates for 'threshold' function	
		Estimated hardware latency	20260600
BINARY	im0	SW-only (Measured cycles)	341040758
		Estimated speedup	16,833
	mountain	SW-only (Measured cycles)	339484132
		Estimated speedup	16,756
	fox	SW-only (Measured cycles)	151631056
		Estimated speedup	7,484
RANGE	im0	SW-only (Measured cycles)	344751442
		Estimated speedup	17,016
	mountain	SW-only (Measured cycles)	343963086
		Estimated speedup	16,977
	fox	SW-only (Measured cycles)	153315602
		Estimated speedup	7,567

Figura 5.4.19: Rendimiento estimado algoritmo Threshold

	Resource utilization estimates for hardware accelerators			
	RESOURCE	USED	TOTAL	UTILIZATION (%)
	DSP	0	80	0
	BRAM	8	60	13,333
	LUT	5053	17600	28,710
	FF	5943	35200	16,884
	HW accelerated (cycles)			Speedup
BINARY	im0	13555550		25,159
	mountain	13550796		25,053
	fox	6050620		56,107
RANGE	im0	13548472		11,192
	mountain	13554070		11,187
	fox	6048762		56,995

Figura 5.4.20: Recursos aceleración hardware Threshold

Debido al gran tiempo de espera para pasar a hardware todos los diseños del proyecto, alguno de ellos no ha sido posible evaluarlos, ya que el tiempo que se emplea entre cada simulación oscila entre los 30 y 45 minutos aproximadamente.

Toda la documentación e información acerca de las funciones de la librería dinámica xfOpenCV se puede consultar en [18].

Las imágenes obtenidas de los algoritmos para el procesamiento de imágenes se encuentran recogidas en el Anexo **B**.

5.5 Comparativa

Como se ha comentado, para visualizar mejor los resultados obtenidos en la placa *Zybo* y verlos de una manera más gráfica, se han recogido en una misma tabla todos los recursos consumidos para cada una de las funciones analizadas (Tabla 5.5.1).

		RESOURCES			
	FUNCTION	**DSP**	**BRAM**	**LUT**	**FF**
FILTERS	gaussian filter	16	10	11096	10031
	sobel filter	0	17	10250	12048
	bilateral filter	19	11	13773	12120
	median blur filter	0	9	6857	7170
	scharr filter	0	17	10582	12253
	dilate	0	9	6000	6726
	erode	0	9	6000	6726
COMPUTATIONS	madd & mult	32	48	17265	23951
	accumulate	0	16	9279	11532
	accumulate squared	1	16	9342	11555
	accumulate weighted	5	16	10142	12201
	phase	4	19	11122	13329
	magnitude	0	16	9657	11631
INPUT PROCESSING	remap	13	36	**18095**	20189
	resize	48	23	16509	30415
	lut	0	16	9858	11642
	channel extract	6	8	5735	6241
OTHER	canny	1	15	11245	9345
	lknpyroflow	35	**106**	**24645**	21675
	pyrDown	1	10	**17734**	8394
	pyrUp	0	13	**17692**	8377
	fast	0	21	16986	19213
	harris	12	32	17453	16774
	warpAffine	**341**	**110**	**33083**	30519
	warpPerspective	**527**	**129**	**39604**	**39099**
	warpTransform	38	**68**	**25123**	19737
	threshold	0	8	5053	5943
	AVAILABLE	**80**	**60**	**17600**	**35200**

Tabla 5.5.1: Recursos consumidos biblioteca xfOpenCV

Los algoritmos de filtrado de imágenes no superan en ningún caso los recursos disponibles en la placa de desarrollo y son posibles llevarlos a hardware. La función bilateral filter emplea un número alto de LUT en comparación con los demás filtros, mientras que dilate y erode consumen el menor número de recursos debido a que no requiere mucha demanda para realizar la comparación de píxeles en la imagen. Estos resultados indican que todavía es posible incluir nuevas funcionalidades para mejorar la arquitectura y los algoritmos

desarrollados sin tener que emplear un dispositivo de gama superior y, por tanto, de mayor coste.

Los recursos consumidos por los algoritmos computacionales y de procesamiento de entrada que se encargan de aplicar un cambio en la imagen original también son adecuados, ya que el área de recursos sigue siendo apropiada para los recursos existentes en la placa. Excepto la función remap que sobrepasa estos límites debido a que hace un uso excesivo de LUT.

En cambio, las funciones que tratan de calcular mediante vectores de movimiento el desplazamiento aparente de un objeto en particular entre dos frames sucesivas, conlleva un alto coste computacional que la placa de desarrollo no puede asumir en hardware. La detección de bordes y esquinas es un proceso altamente costoso en hardware, al igual que la aplicación de una operación de rotación de imagen, escalado y transformación de perspectiva que conllevan un gran coste de recursos.

5.6 Cámara Pcam 5C

El procesamiento de vídeo con el componente Pcam 5C instalado en la placa de desarrollo *Zybo Z7* ha sido una tarea difícil de ejecutar. Se han producido numerosos problemas a la hora de desarrollar la demo del módulo *Zybo Z7 Pcam 5C.*

El principal problema ha sido la incompatibilidad entre versiones. En versiones más recientes y en concreto para la usada en este proyecto (*SDSoC 2017.2*) no existe actualmente dicho soporte. Por ello, ha sido necesaria la instalación de *Vivado 2016.4* y la herramienta *SDK 2016.4* en la que se han llevado a cabo pruebas, aunque la documentación sigue siendo muy escasa.

Sin embargo, ha sido posible completar su síntesis e implementación, así como la generación del *bitstream* de datos para posteriormente programar la FPGA.

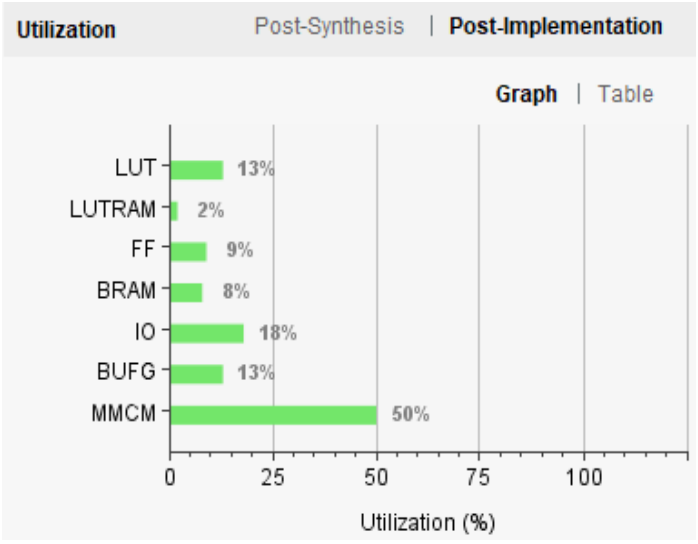


Figura 5.6.1: Utilización de recursos totales Pcam 5C

La utilización de los recursos disponibles (Figura 5.6.1) es relativamente baja para el total que posee la placa de desarrollo, salvo la ocupación de MMCM que se sitúa en el 50%.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Los objetivos marcados al principio de esta memoria fueron los siguientes:

1. Estudio de la herramienta *SDSoC* y placas de desarrollo *Zybo* / *Zybo Z7*.
2. Síntesis e implementación de funciones software como asistidas por hardware.
3. Análisis de latencias, rendimiento en cuanto a tiempo de ejecución y recursos consumidos.

El primer objetivo, descrito en el capítulo 2, se ha completado con éxito, ya que una vez aprendidos los conocimientos necesarios de la herramienta, ha sido posible desarrollar diferentes diseños de referencia gracias a las funciones de las librerías dinámicas OpenCV y xfOpenCV. También ha sido realmente importante examinar las características y limitaciones que presentaban las placas de desarrollo *Zybo* y *Zybo Z7* para el análisis de las mediciones.

El segundo objetivo, cubierto en el capítulo 3 para el procesamiento de imágenes, también se ha llevado a cabo satisfactoriamente (Tabla 5.5.1), aunque en un primer momento surgieron diversos contratiempos que dificultaron su puesta en marcha. Sin embargo, para el procesamiento de vídeo explicado en el capítulo 4, no se ha llegado a finalizar por completo el objetivo principal que era hacer funcionar la cámara de vídeo a tiempo real con baja latencia.

La causa principal como ya se ha comentado, ha sido la incompatibilidad de ciertas versiones *SDSoC* con las librerías dinámicas xfOpenCV y la carencia de información que a día de hoy es muy limitada, puesto que son dispositivos que están en pleno desarrollo y con un potencial aún por explotar.

Después de solucionar estos problemas, se han conseguido realizar distintos diseños software y posteriormente en versión hardware en ambas placas de desarrollo que se han utilizado durante el proyecto. En efecto, estos resultados en cuanto a rendimiento y recursos empleados se encuentran recogidos en el capítulo 5.

También ha sido beneficiosa la asistencia a un seminario celebrado y organizado en la Universidad Autónoma de Madrid, en el que la empresa de productos de ingeniería eléctrica Digilent enseñaba un curso breve de *Vivado HLS* [4] para que los estudiantes interesados adquirieran nuevos conocimientos acerca de la herramienta de desarrollo de sistemas empotrados.

6.1.1 Sobre SDSoC

A lo largo de este proyecto han surgido diversos problemas con el entorno de desarrollo *SDSoC*. En primera instancia, cabe destacar que la utilización de librerías dinámicas resulta de gran ayuda para el diseño e implementación de sistemas empotrados. La herramienta no tiene incluida por defecto la librería dinámica xfOpenCV ni la biblioteca OpenCV que dan soporte para el procesamiento de imágenes y por ello, ha sido necesario completar sus instalaciones correspondientes para hacer uso de ellas.
En segundo lugar, la portabilidad entre distintas versiones de la herramienta también ha sido un aspecto negativo a tener en cuenta. Durante el tiempo empleado en la realización de este Trabajo Fin de Grado, Xilinx ha lanzado al mercado nuevas actualizaciones del entorno (2017.4 y 2018.1). Sin embargo, al intentar trasladar el proyecto se han producido errores de ejecución inesperados por parte de la herramienta que han generado resultados indeseados, lo que ha resultado muy complicado conformarlo en estas versiones. Esto se debe a la falta de semejanza entre las versiones existentes.

En consecuencia, la automatización del entorno de desarrollo *SDSoC* para desarrolladores de código software que carecen de conocimientos hardware, aún no está suficientemente extendido y es posible que en el futuro se consiga, pero a día de hoy resulta muy incierto. En general, el aprendizaje personal mediante el método de prueba y error de los aspectos aún no documentados ha sido fundamental para el desarrollo de los objetivos principales.

De todas maneras, se puede determinar que *SDSoC* es una herramienta con gran utilidad que cumple las funciones para la cual fue creada, ya que resulta de gran ayuda para sintetizar e implementar diseños software compilados con funciones aceleradas para procesadores ARM y posteriormente montarlo en placas FPGA. Esto hubiera resultado más complejo con el uso de otros entornos con síntesis de bajo nivel, aunque con mayor experiencia en conocimientos de funcionamiento.

6.2 Trabajo futuro

A la vista de los resultados que se han obtenido en este trabajo se propone trabajar sobre nuevas metodologías de diseño, así como combinar diferentes herramientas, nuevas librerías de detección y procesamiento de imágenes, etc… con el objetivo de reducir el esfuerzo de diseño, implementación y testing.

Otro aspecto a tratar podría ser la síntesis, implementación, optimización y aceleración de hardware en otro entorno de desarrollo y comparar la empleabilidad de ambas aplicaciones para observar cual es la más adecuada en cuanto a prestaciones, rendimiento y consumo de energía. De este modo, se podría evaluar completamente las numerosas ventajas que supone la herramienta.

Como se comentó en la sección 6.1, existen dudas sobre la incompatibilidad y la capacidad de adaptación en las recientes actualizaciones que van saliendo a la luz con el paso del tiempo. Por lo tanto, en trabajos futuros se plantea la alternativa de conseguir la portabilidad de los diseños desarrollados incluyendo las librerías dinámicas en versiones posteriores.

7 Referencias

[1] Alekhin, A. (n.d.). *OpenCV: Open Source Computer Vision Library*. [online] GitHub. Available at: https://github.com/opencv/opencv

[2] Bobrowicz, S. (n.d.). *Zybo Z7-20 SDSoC Platform*. [online] GitHub. Available at: https://github.com/Digilent/SDSoC-Zybo-Z7-20

[3] Borra, G. (n.d.). *Xilinx xfOpenCV Library*. [online] GitHub. Available at: https://github.com/Xilinx/xfopencv

[4] Digilent (2017). *Seminario ZYBO Z7 Video Workshop*

[5] Digilent (2017). *Zybo*. [online] Available at: https://reference.digilentinc.com/reference/programmable-logic/zybo/start

[6] Digilent (2017). *Zybo Reference Manual*. [online] Available at: https://reference.digilentinc.com/reference/programmable-logic/zybo/reference-manual

[7] Digilent (2018). *Pcam 5C*. [online] Available at: https://reference.digilentinc.com/reference/add-ons/pcam-5c/start

[8] Digilent (2018). *Pcam 5C Reference Manua*l. [online] Available at: https://reference.digilentinc.com/reference/add-ons/pcam-5c/reference-manual

[9] Digilent (2018). *Zybo Z7*. [online] Available at: https://reference.digilentinc.com/reference/programmable-logic/zybo-z7/start

[10] Digilent (2018). *Zybo Z7 Migration Guide*. [online] Available at: https://reference.digilentinc.com/reference/programmable-logic/zybo-z7/migration-guide

[11] Digilent (2018). *Zybo Z7 Pcam 5C Demo*. [online] Available at: https://reference.digilentinc.com/learn/programmable-logic/tutorials/zybo-z7-pcam-5c-demo/start

[12] Digilent (2018). *Zybo Z7 Reference Manual*. [online] Available at: https://reference.digilentinc.com/reference/programmable-logic/zybo-z7/reference-manual

[13] Gyorgy, E. (n.d.). *Zybo Z7-20 Pcam 5C Demo Project*. [online] GitHub. Available at: https://github.com/Digilent/Zybo-Z7-20-pcam-5c

[14] OpenCV Documentation. [online] Available at: https://docs.opencv.org/

[15] OpenCV Module 3.4.0 [online] Available at: https://docs.opencv.org/3.4.0/

[16] Xilinx (2017). *SDSoC Environment User Guide (UG1027)*. [online] Available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug1027-sdsoc-user-guide.pdf

[17] Xilinx (2017). *SDSoC Environment User Guide: Introduction (UG1028)*. [online] Available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug1028-sdsoc-intro-tutorial.pdf

[18] Xilinx (2017). *Xilinx OpenCV User Guide (UG1233)*. [online] Available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug1233-xilinx-opencv-user-guide.pdf

8 Glosario

ARM	Advanced RISC Machines
AWB	Automatic White Balance
BRAM	Block RAM
DSP	Digital Signal Processing
FF	Flip-Flop
FPGA	Field Programmable Gate Array
fps	frames per second
HLS	High Level Synthesis
IP	Intelectual Property
JTAG	Joint Test Action Group Boundary-scan
LUT	Look Up Table
MMCM	Mixed-Mode Clock Manager
OpenCV	Open Computer Vision
SDSoC	Software Defined System-on-Chip
SoC	System-on-Chip
SSCB	Serial Camera Control Bus
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus

Anexos

A Manual de implementación de software a hardware

Para llevar a cabo la implementación del programa software y montarlo en hardware es necesario poseer una tarjeta microSD en la que se guardarán los archivos implementados. Sin embargo, previamente en esta tarjeta ha sido necesario montar una partición contigua con formato FAT32 debido a que el entorno de desarrollo no traía por defecto las librerías dinámicas de OpenCV que permiten enlazar con el ejecutable .elf. En una de estas particiones (BOOT) se almacenarán los archivos que van a ser ejecutados en la FPGA, mientras que en la otra partición (libs) se encuentran todas las librerías obligatorias que se usarán para compilar y generar los diseños.

mkdir /lib/user_libs
mount /dev/mmcblk0p2 /lib/user_libs
export LD_LIBRARY_PATH=/lib/user_libs/opencv/

En primer lugar, se abre la plataforma de desarrollo *SDx 2017.2* y se carga el diseño con el que vamos a trabajar. Para ello, anteriormente es necesario indicar la placa de desarrollo con la que vamos a generar el diseño del proyecto, así como el sistema operativo y configuración del sistema.

Una vez completados estos cambios, se debe cambiar la configuración de la herramienta GNU para que busque las librerías dinámicas en los sitios correspondientes a la hora de construir y compilar el proyecto. Se hace clic con el botón derecho encima del proyecto y a continuación *C/C++ Build Settings.* Hay que añadir las rutas de las cabeceras include de las librerías OpenCV y xfOpenCV en el directorio *SDS++ Compiler*.

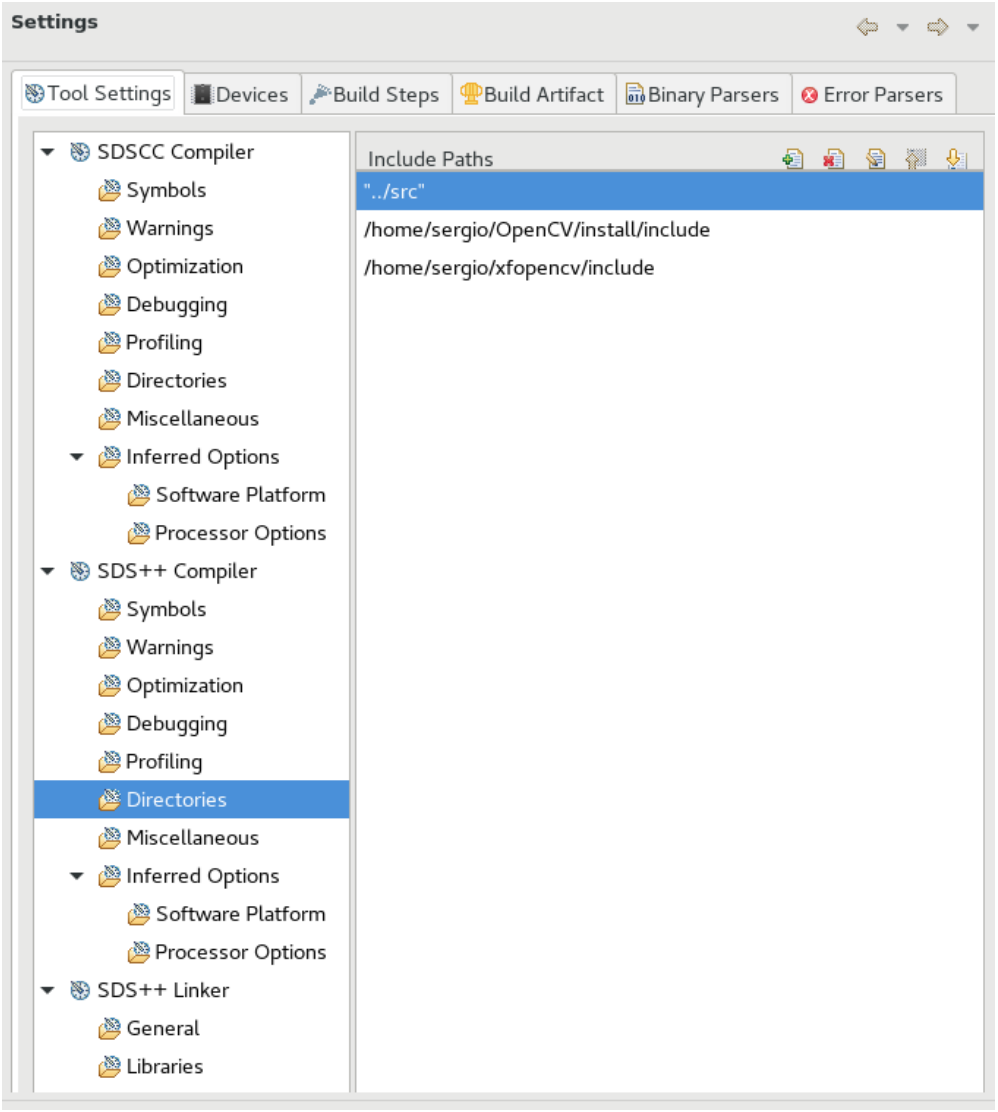


Figura A.1: Directorio *SDS++ Compiler*

Además, también es imprescindible enlazar las librerías de *SDS++ Linker*: opencv_core, opencv_flann, opencv_imgcodecs, opencv_imgproc, opencv_features2d, opencv_calib3d y el directorio lib de las librerías dinámicas de OpenCV.

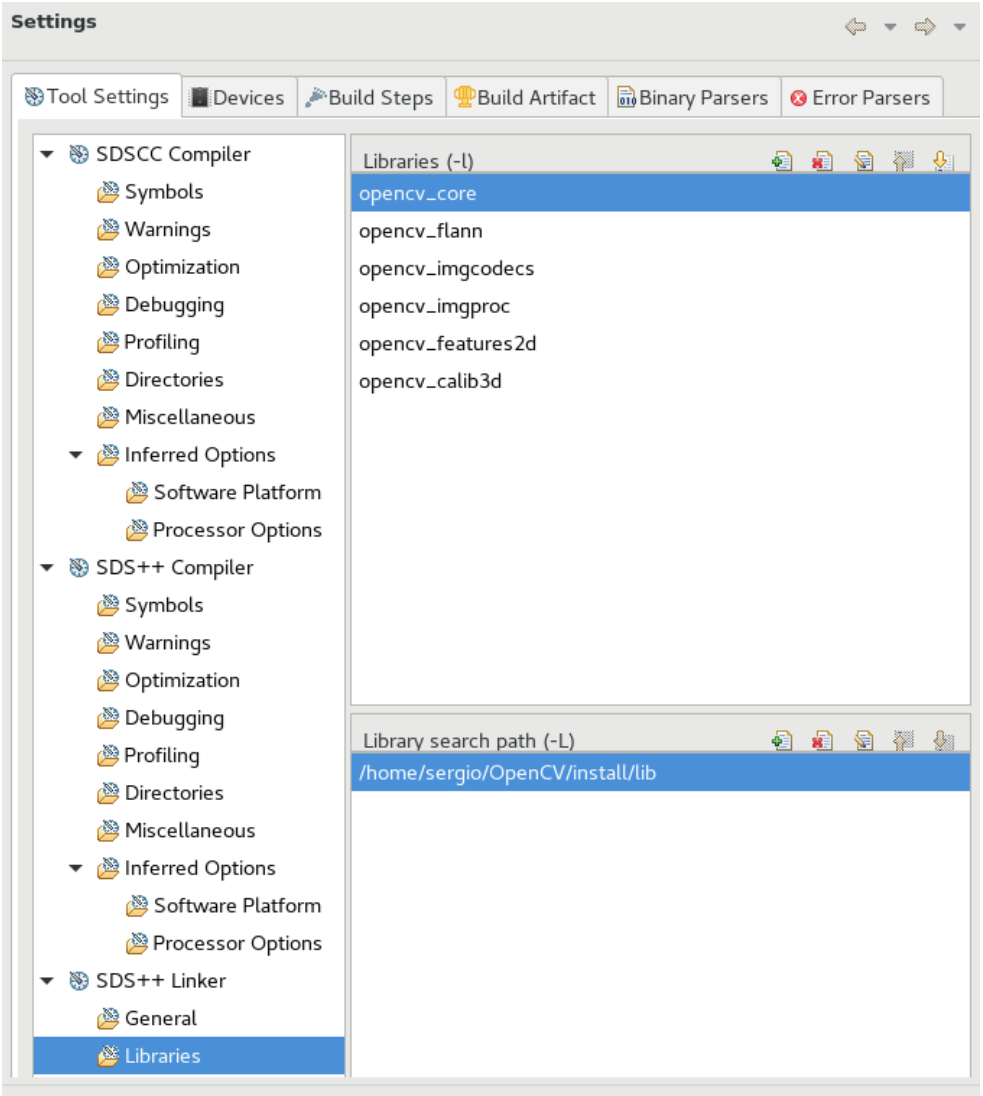


Figura A.2: Directorio *SDS++ Linker*

Se hace clic derecho encima de la función de la biblioteca que se desea mover a hardware y se pulsa en la opción "*Toggle HW/SW*" para convertirla a hardware.

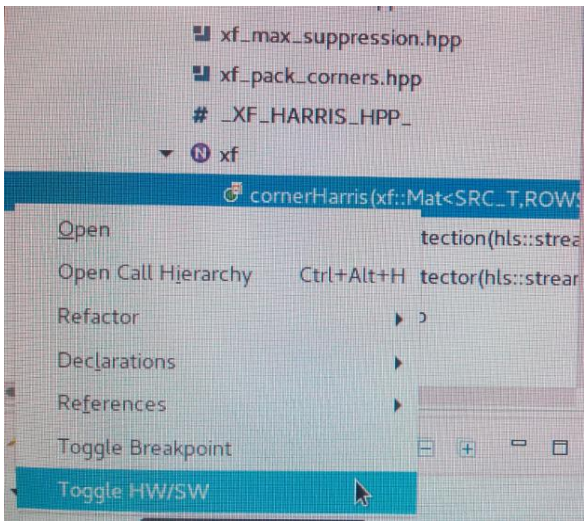


Figura A.3: Toggle HW/SW

Finalmente, se construye el proyecto con el botón martillo *Build* situado en la parte superior del programa. Este proceso de construcción puede tardar entre 30-45 minutos, ya que emplea tiempo para la síntesis e implementación, optimización y colocación de la lógica, enrutamiento en el dispositivo y, por último, el bitstream, es decir, la generación del flujo de bits en la FPGA.

```
Finished 1st of 5 tasks (FPGA synthesis). Elapsed time: 00h 31m 13s.

Finished 2nd of 5 tasks (FPGA logic optimization). Elapsed time: 00h 00m 30s.

Finished 3rd of 5 tasks (FPGA logic placement). Elapsed time: 00h 00m 56s.

Finished 4th of 5 tasks (FPGA routing). Elapsed time: 00h 00m 41s.

Finished 5th of 5 tasks (FPGA bitstream generation). Elapsed time: 00h 01m 01s.
```

Figura A.4: Síntesis e implementación

Posteriormente, al finalizar por completo la compilación sin ningún error, buscamos en nuestro directorio de trabajo una carpeta llamada *Debug* o *Release*. Para obtener un mejor rendimiento en tiempo de ejecución, la configuración elegida debe ser *Release*, ya que usa una optimización de compilador mayor que la configuración de depuración *Debug*.

Dentro habrá una carpeta llamada *sd_card* que será aquella que copiemos todo el contenido a la tarjeta microSD. Se extrae del host y posteriormente se inserta en la placa de desarrollo. Hará falta la conexión del puerto USB UART de la placa al ordenador a través de un cable USB y la conexión de una fuente de alimentación externa adecuada para alimentar el dispositivo.

Por último, se acciona el interruptor para encender la placa y se abre una terminal de comandos en el ordenador. Primero es necesario activar los privilegios de administrador (*sudo su*) y luego se conecta al dispositivo ttyUSB1 a 115200 baudios, sin control de flujo y sin paridad (*screen /dev/ttyUSB1 115200*). Una vez que se ha conectado satisfactoriamente, se ejecutan los siguientes comandos en la terminal:

cd /mnt
./<project_name>.elf (ficheros necesarios)

donde project_name es el nombre del ejecutable creado y a continuación hará falta nombrar las imágenes que van a ser procesadas.

B Imágenes resultantes

❖ ***Filtros***

- **Gaussian Filter**

Figura B.1: Resultado *im0* hardware Gaussian Filter

- **Sobel Filter**

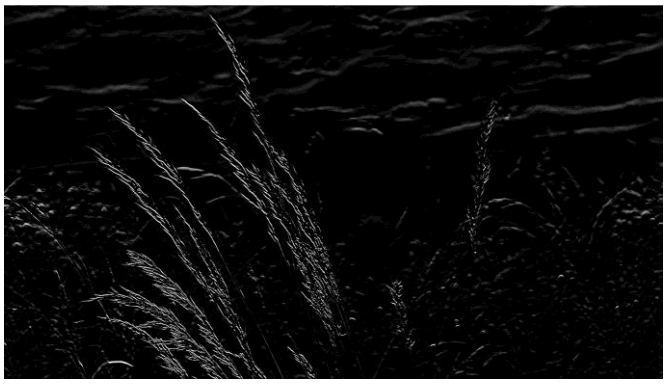


Figura B.2: Resultado *im0* hardware Sobel Filter, dirección x, y

Figura B.3: Resultado *lenna* hardware Sobel Filter, dirección x, y

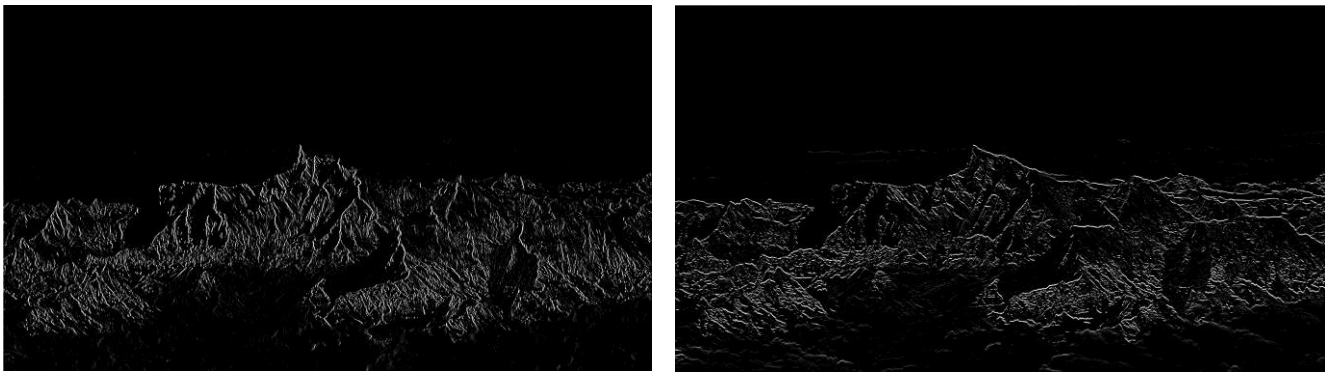


Figura B.4: Resultado *mountain* hardware Sobel Filter, dirección x, y

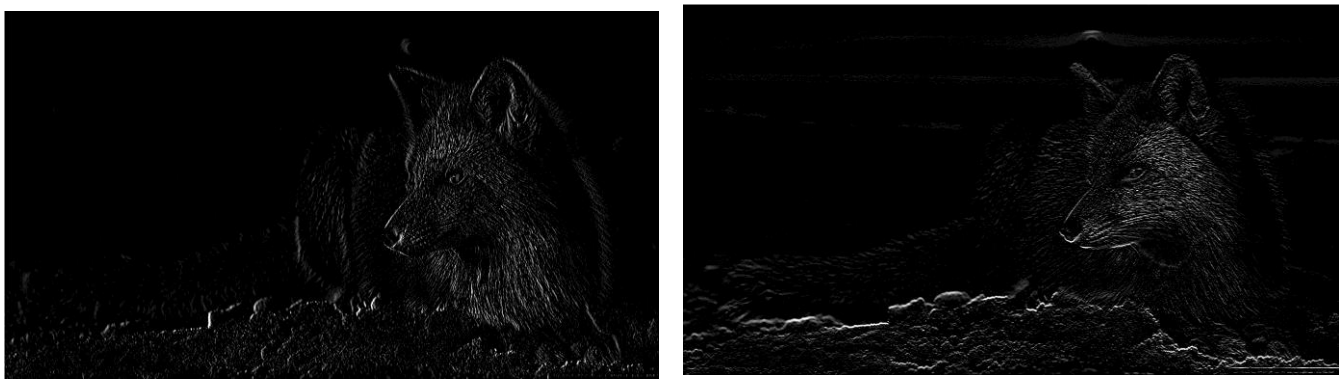


Figura B.5: Resultado *fox* hardware Sobel Filter, dirección x, y

- **Bilateral Filter**

Figura B.6: Resultado *im0* hardware Bilateral Filter

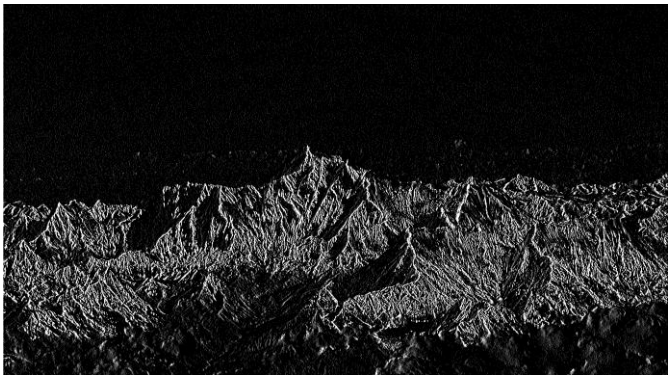
Figura B.7: Resultado *mountain* hardware Bilateral Filter

- **Median Blur Filter**

Figura B.8: Resultado *im0* hardware Median Blur Filter

- **Scharr Filter**

Figura B.9: Resultado *im0* hardware Scharr Filter, dirección x, y



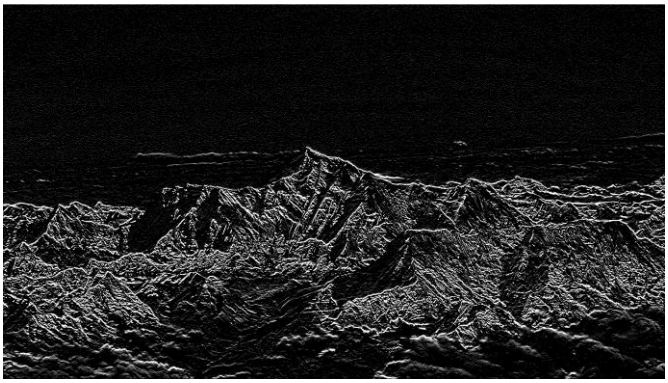


Figura B.10: Resultado *mountain* hardware Scharr Filter, dirección x, y

- **Dilate**

Figura B.11: Resultado *im0* hardware Dilate

- **Erode**

Figura B.12: Resultado *im0* hardware Erode

Figura B.13: Resultado *mountain* hardware Erode

❖ ***Cálculos***

- **Accumulate**

Figura B.14: Resultado *im0+mountain* hardware Accumulate

- **Accumulate Squared**

Figura B.15: Resultado *im0+im1* hardware Accumulate Squared

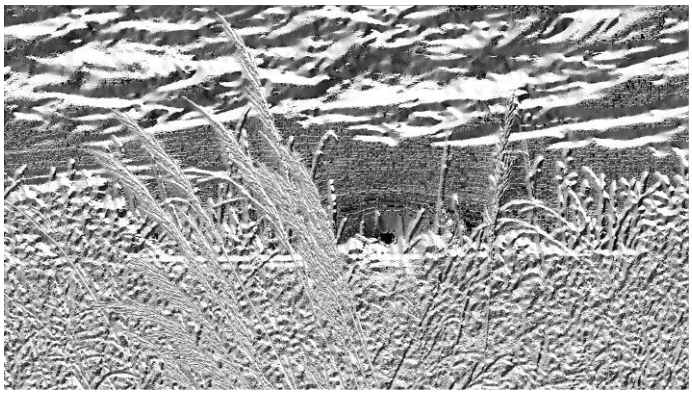
Figura B.16: Resultado *im0+mountain* hardware Accumulate Squared

- **Accumulate Weighted**

Figura B.17: Resultado *im0+im1* hardware Accumulate Weighted

Figura B.18: Resultado *im0+mountain* hardware Accumulate Weighted

- **Phase**



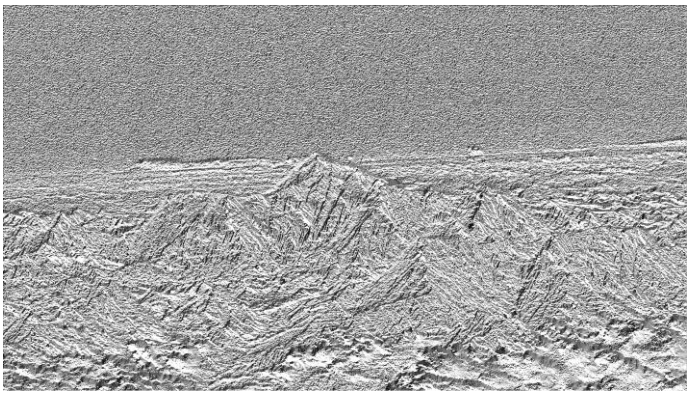


Figura B.19: Resultado *im0* hardware Phase

Figura B.20: Resultado *mountain* hardware Phase

- **Magnitude**

Figura B.21: Resultado *im0* hardware Magnitude

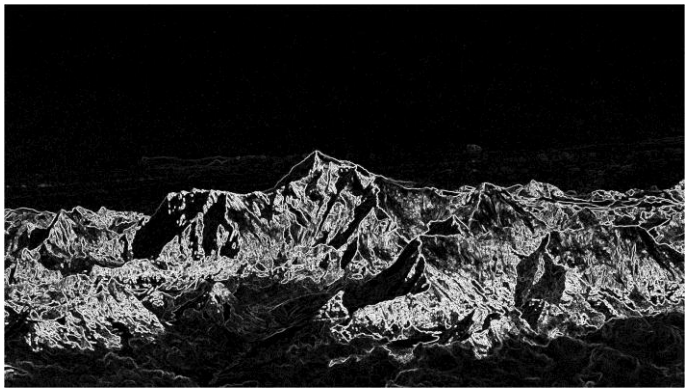


Figura B.22: Resultado *mountain* hardware Magnitude

❖ ***Procesamiento de entrada***

- **Remap**

No es posible llevarla a hardware debido que hace un uso excesivo de recursos LUT. Sin embargo, este es el resultado en software. La imagen se ha invertido como si se tratase de un espejo.

Figura B.23: Resultado *im0* software Remap

Figura B.24: Resultado *mountain* software Remap

- **Resize**

Figura B.25: Resultado *im0* interp área hardware Resize

Figura B.26: Resultado *im0* interp bilineal hardware Resize

Figura B.27: Resultado *im0* interp vecino hardware Resize

Las dimensiones de las imágenes se han visto reducidas a 640x480 píxeles.

- **Channel Extract**

Figura B.28: Resultado *im0* hardware Channel Extract

Figura B.29: Resultado *mountain* hardware Channel Extract

❖ ***Otros***

➢ Canny

Figura B.30: Resultado *im0* hardware Canny

Figura B.31: Resultado *lenna* hardware Canny

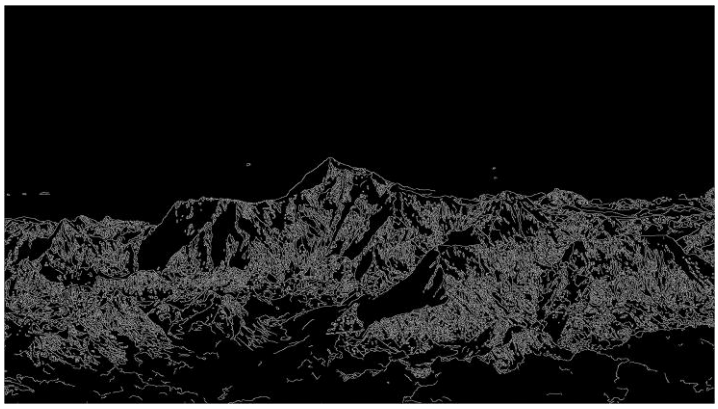


Figura B.32: Resultado *fox* hardware Canny

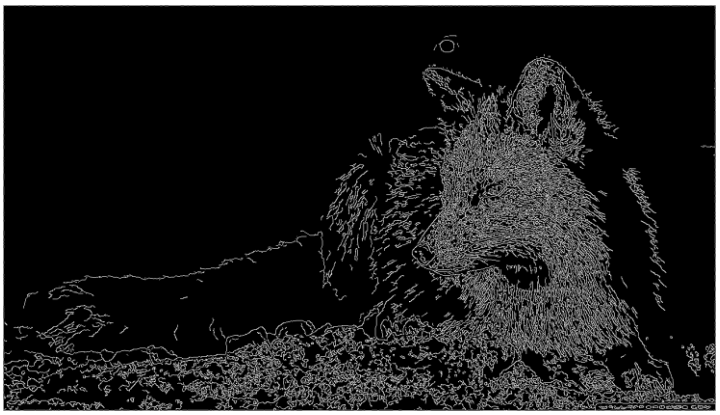


Figura B.33: Resultado *mountain* hardware Canny

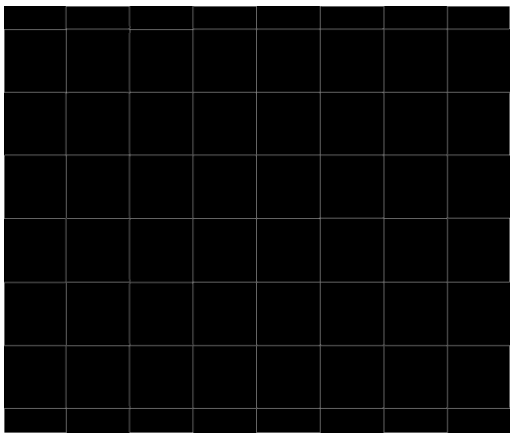


Figura B.34: Resultado *square* hardware Canny

- **Lknpyroflow**

No es posible llevarla a hardware debido que hace un uso excesivo de recursos BRAM y LUT. Sin embargo, este es el resultado en software:

Figura B.35: Resultado *car* software Lknpyroflow

- **PyrDown**

No es posible llevarla a hardware debido que hace un uso excesivo de recursos LUT. Sin embargo, este es el resultado en software:

Figura B.36: Resultado *im0* software PyrDown

Figura B.37: Resultado *mountain* software PyrDown

Las imágenes se han visto reducidas a la mitad, es decir, de 1920x1080 a 960x540 píxeles.

- **PyrUp**

No es posible llevarla a hardware debido que hace un uso excesivo de recursos LUT. Sin embargo, este es el resultado en software:

Figura B.38: Resultado *im0* software PyrUp

Figura B.39: Resultado *mountain* software PyrUp

- **Fast**

Figura B.40: Resultado *im0* hardware Fast

Figura B.41: Resultado *mountain* hardware Fast

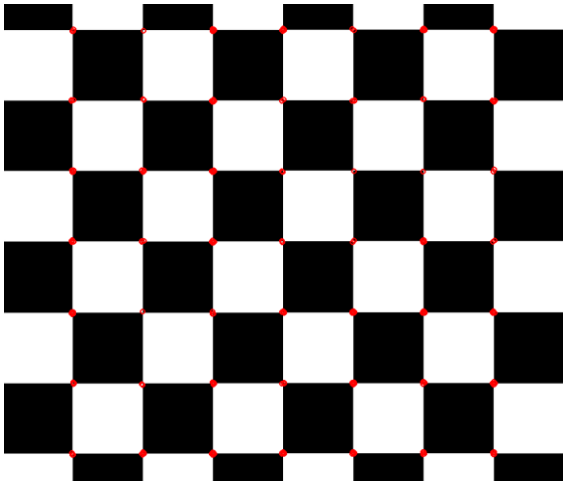


Figura B.42: Resultado *square* hardware Fast

- **Harris**

Figura B.43: Resultado *im0* hardware Harris

Figura B.44: Resultado *mountain* hardware Harris

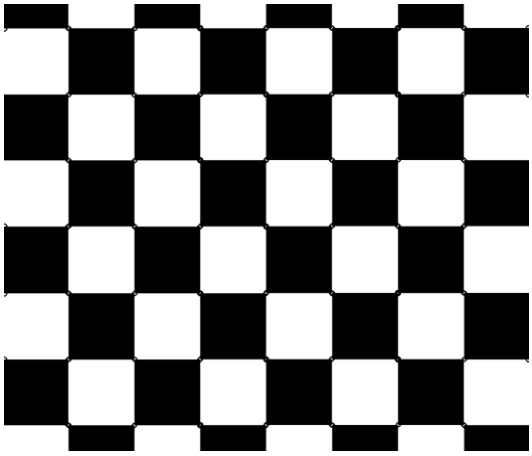


Figura B.45: Resultado *square* hardware Harris

- **WarpAffine**

No es posible llevarla a hardware debido que hace un uso excesivo de recursos DSP, BRAM y LUT. Sin embargo, este es el resultado en software:

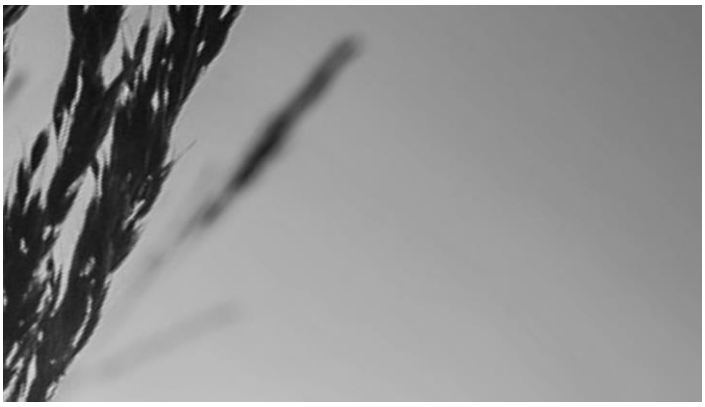


Figura B.46: Resultado *im0* software WarpAffine

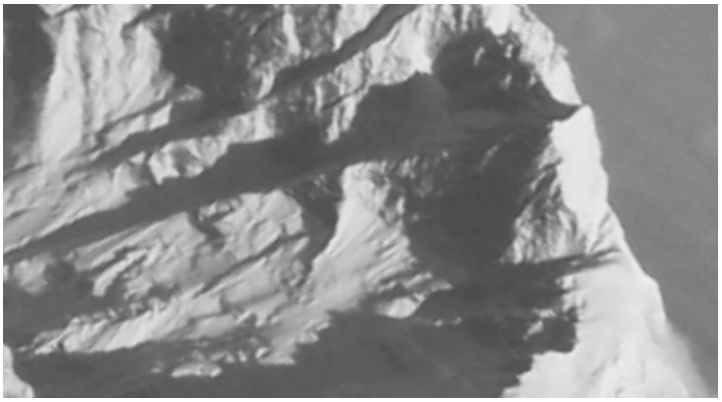


Figura B.47: Resultado *mountain* software WarpAffine

- **WarpPerspective**

No es posible llevarla a hardware debido que hace un uso excesivo de recursos DSP, BRAM, LUT y FF. Sin embargo, este es el resultado en software:

Figura B.48: Resultado *im0* software WarpPerspective

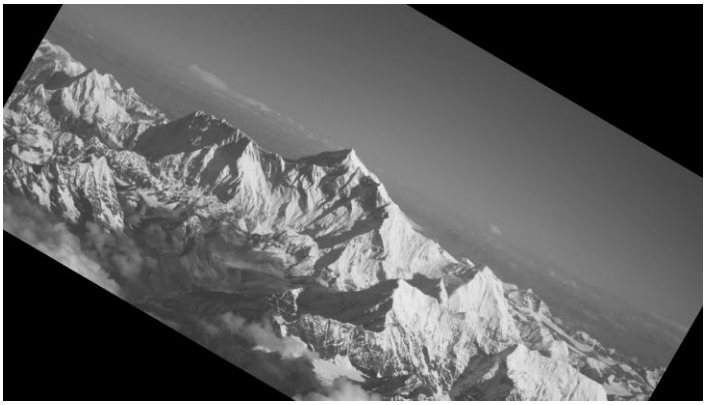


Figura B.49: Resultado *mountain* software WarpPerspective

- **WarpTransform**

No es posible llevarla a hardware debido que hace un uso excesivo de recursos BRAM y LUT. Sin embargo, este es el resultado en software:

Figura B.50: Resultado *im0* software WarpTransform

- **Threshold**

Figura B.51: Resultado *im0* hardware binary Threshold

Figura B.52: Resultado *fox* hardware binary Threshold

Figura B.53: Resultado *im0* hardware range Threshold

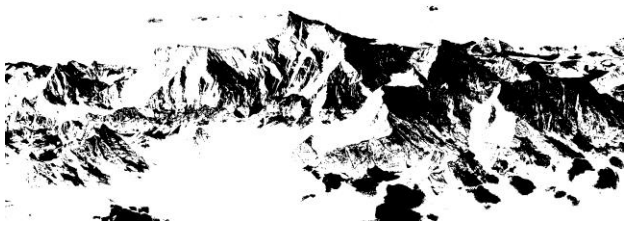


Figura B.54: Resultado *mountain* hardware range Threshold

Figura B.55: Resultado *fox* hardware range Threshold